VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SIMULÁTOR TURINGOVÝCH STROJŮ POPSANÝCH POMOCÍ KOMPOZITNÍCH DIAGRAMŮ

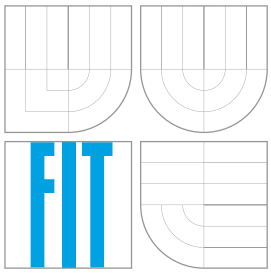
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JOSEF SISKA

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SIMULÁTOR TURINGOVÝCH STROJŮ POPSANÝCH POMOCÍ KOMPOZITNÍCH DIAGRAMŮ

SIMULATOR OF TURING MACHINES DESCRIBED BY MEANS OF COMPOSITE DIAGRAMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JOSEF SISKA

VEDOUCÍ PRÁCE
SUPERVISOR

Mgr. ADAM ROGALEWICZ, Ph.D.

BRNO 2014

Abstrakt

V této práci je uvedena teorie související s Turingovými stroji a formami jejich popisu se zaměřením na kompozitní diagramy. Cílem práce je vytvořit aplikaci, která umožní editaci Turingových strojů zapsaných pomocí kompozitních diagramů a simulaci jejich běhu na zadané vstupní konfiguraci (včetně strojů nedeterministických i vícepáskových). Dále bude aplikace umožňovat spustit analýzu daného Turingova stroje za účelem zjištění, zda tento stroj nebo některé jeho části vždy zastaví. Výsledná aplikace poskytující uvedené funkce je implementována v Javě a zmíněná analýza je v ní prováděna s využitím konstrukce fundovaných uspořádání. V rámci práce tak vznikl nástroj umožňující návrh a testování Turingových strojů zapsaných pomocí kompozitních diagramů. Aplikace může najít své využití zejména při výuce teoretické informatiky, kde může posloužit např. pro demonstraci činnosti daného Turingova stroje.

Abstract

In this thesis, the theory related to Turing machines and means of their description (with focus on composite diagrams) is presented. The aim of this work is to create an application that allows editing Turing machines described by means of composite diagrams and simulating their computation on specified input configuration (including non-deterministic and multi-tape machines). Furthermore, within the application it will be possible to run the termination analysis of Turing machine in order to determine whether this machine or any of its parts always halt. The resulting application is implemented in Java and the termination analysis is performed using the well-founded orders. And so, one of the results created during this work is a software tool which allows designing and testing of Turing machines described by means of composite diagrams. Resulting application may be used especially during lectures on theoretical computer science, where it can be used to demonstrate computation of some Turing machine.

Klíčová slova

Turingův stroj, kompozitní diagram, simulace, problém zastavení, fundované uspořádání

Keywords

Turing machine, composite diagram, simulation, halting problem, well-founded order

Citace

Josef Siska: Simulátor Turingových strojů popsaných pomocí kompozitních diagramů, diplomová práce, Brno, FIT VUT v Brně, 2014

Simulátor Turingových strojů popsaných pomocí kompozitních diagramů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Mgr. Adama Rogalewicze, Ph.D. a že jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Josef Siska
15. května 2014

Poděkování

Rád bych zde poděkoval vedoucímu diplomové práce Mgr. Adamu Rogalewiczovi, Ph.D. za cenné připomínky a rady poskytnuté během vypracovávání diplomové práce.

Obsah

Kapitola 1

Úvod

Počítače jsou v dnešní době nedílnou součástí našeho života. Slouží mnoha odlišným účelům a vyskytují se snad v každém odvětví lidské činnosti, jako příklad si uveďme letectví, automobilový průmysl, telekomunikace apod. Přestože lze za pomoci počítačů řešit širokou škálu problémů, nelze takto řešit všechny. Pro určení hranic toho, co lze na počítačích řešit se často využívají tzv. Turingovy stroje, kterými se v této práci budu detailně zabývat. Turingův stroj je jednoduchým teoretickým modelem počítače navrženým Alanem Turingem v roce 1936 ([1]). Turingovými stroji se zabývá obor informatiky zvaný teoretická informatika, konkrétně oblast soustředěná na automaty a gramatiky. Důležitost Turingova stroje v teoretické informatice a v informatice obecně tkví v tom, že není znám efektivně vyčíslitelný proces, který by nebylo možné popsat Turingovým strojem (viz tzv. Churchova teze v kapitole 2.2), ale také v tom, že na jeho základě je obvykle definován pojem algoritmus. Platí, že vše, co lze řešit na reálném počítači, je možné řešit na Turingově stroji a naopak ([2, 3]).

Funkci určitého daného Turingova stroje lze zachytit několika způsoby, z nichž některé si v této práci blíže popíšeme, přičemž nejvíce se budeme věnovat popisu pomocí kompozitních diagramů. V práci se také zmíníme o problému zastavení Turingova stroje, což je důležitý nerozhodnutelný problém v teoretické informatice, pro který bude představen základní přístup na jeho částečné řešení založený na tzv. fundovaných uspořádáních. Praktická část této práce je soustředěna na vytvoření aplikace s grafickým uživatelským rozhraním, která umožňuje editaci Turingových strojů zapsaných pomocí kompozitních diagramů a simulaci jejich běhu na zadané vstupní konfiguraci. Součástí aplikace je také heuristika provádějící analýzu zadaného kompozitního diagramu Turingova stroje za účelem zjištění, zda tento stroj vždy zastaví příp. zda nejsme schopni odpovědět.

Obsah této práce je rozdělen následovně. Nejprve budou představeny základní definice týkající se Turingových strojů a bude popsán princip jejich fungování. Třetí kapitola je věnována variantám Turingových strojů, což jsou nejrůznějším způsobem modifikované Turingovy stroje. Pro některé z nich zde budou přiblíženy důkazy jejich ekvivalence s klasickým Turingovým strojem. V následující kapitole jsou uvedeny formy popisu Turingových strojů, tedy způsoby jak Turingovy stroje zapisovat, přičemž hlavní část této kapitoly je věnována kompozitním diagramům. Pátá kapitola se zabývá již zmíněným problémem zastavení Turingova stroje, před nímž jsou ale ještě zavedeny pojmy jako rozhodovací problém, kódování Turingova stroje, univerzální Turingův stroj a v závěru kapitoly také fundované uspořádání. Dále si v práci stručně nastíníme problematiku analýzy složitosti výpočtu Turingova stroje. V této kapitole bude také uvedena motivace pro její studium ve formě problémů, na které můžeme narazit při analýze složitosti programů. Kapitoly sedm a osm se už týkají praktické

části práce, konkrétně je zde popsán návrh vytvořené aplikace a její implementace.

Tato diplomová práce navazuje na semestrální projekt „Simulátor Turingových strojů popsaných pomocí kompozitních diagramů“ ([4]) vypracovaný v zimním semestru. Cílem semestrálního projektu bylo jednak zpracování problematiky Turingových strojů, jejich variant a forem popisu, ale také navržení podoby výsledné aplikace. Kapitoly 2, 3, 4 a 7 uvedené v této práci jsou tedy založeny na stejnojmenných kapitolách ze semestrálního projektu.

Kapitola 2

Turingovy stroje

V této kapitole bude nejprve uvedena historie vzniku Turingova stroje, poté krátká motivace pro jeho studium ve formě Churchovy teze a ve zbytku kapitoly bude uveden základní popis a související definice týkající se Turingova stroje. V této kapitole jsem čerpal z [2, 5, 6, 7, 8, 9, 10].

2.1 Historie vzniku Turingova stroje

Na přelomu devatenáctého a dvacátého století předložil světu německý matematik David Hilbert seznam 23 matematických problémů, které byly výzvou pro matematiku 20. století. Jedním z problémů, který sice nebyl součástí tohoto seznamu, ale jistým způsobem s ním souvisel, byl tzv. rozhodovací problém („Entscheidungsproblem“). Ten se zabýval otázkou, zda ke každému matematickému výroku existuje ryze formální, mechanický proces, který by rozhodl, zda je tento výrok pravdivý či nikoliv. Alonzo Church a Alan Turing vytvořili v letech 1935–1936 nezávisle na sobě dvě různá řešení tohoto problému, která dokazují, že takovýto algoritmus nelze nalézt ([7, 10]). Churchovým řešením byl λ-kalkul ([11, 12]), kterým se zde ale nebudeme dále zabývat. Turing pak ve svém článku [1] navrhnul stroj dnes nazývaný jeho jménem – Turingův stroj (dále také jako TS). Spolu s těmito řešeními vzniklo i několik dalších formalismů – Postovy systémy ([13, 14], autorem Emil Post), μ-rekurzivní funkce ([15], autorem Kurt Gödel) a další. Jak se ale později ukázalo, mají všechny tyto formalismy stejnou výpočetní sílu, což mimo jiné vedlo k formulaci tzv. Churchovy teze ([2, 8]).

2.2 Churchova teze

Cílem Turinga v době, kdy definoval svůj model, bylo určit, co je a co není efektivní procedura, resp. co je a co není algoritmus ([6]). Nejprve si proto uveďme, co dnes chápeme pod pojmem algoritmus ([9]):

Algoritmus je přesně definovaná konečná posloupnost příkazů (kroků), jejichž prováděním pro každé přípustné vstupní hodnoty získáme po konečném počtu kroků odpovídající hodnoty výstupní.

Definice algoritmu je tedy neformální, což činí problémy, pokud chceme formálně dokázat, že neexistuje žádný algoritmus pro řešení určitého problému. To vedlo k několika

dalším definicím pojmu algoritmus, z nichž jedna vychází z Turingova stroje. Konkrétně jde o tzv. Churchovu tezi (někdy také Church-Turingova teze):

Každý proces, který lze intuitivně nazvat algoritmem, se dá realizovat na Turingově stroji.

Jinak také formulována jako ([5]):

Turingovy stroje (a jim ekvivalentní systémy) definují svou výpočetní silou to, co intuitivně považujeme za efektivně vyčíslitelné.

Obsahem Churchovy teze je tedy ztotožnění pojmů „algoritmicky řešitelný“ a „řešitelný Turingovým strojem“. Pojem „algoritmicky řešitelný“ chápeme pouze jako intuitivní pojem a tedy obsah Churchovy teze nemůže být formálně dokázán. Existuje však celá řada argumentů podporujících platnost této teze:

- Turingovy stroje jsou velmi robustní – jejich různá omezení ale i rozšíření nemají žádný vliv na jejich výpočetní sílu (viz kapitola 3).
- Kromě Turingových strojů bylo navrženo i mnoho jiných formalismů (viz výše) se shodnou výpočetní silou jako TS.
- Doposud není znám žádný algoritmus, který by nebylo možné realizovat na Turingově stroji.

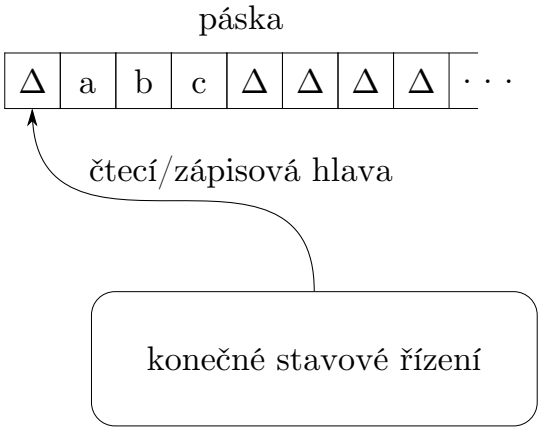
Churchova teze je všeobecně uznávaná jako pravdivá, což vede k tomu, že bývá obrácena a použita jako definice algoritmu (výpočetního postupu):

(Obrácená Churchova teze). *Algoritmem (výpočetním postupem) je právě to, co lze realizovat Turingovým strojem.*

2.3 Neformální popis Turingova stroje

Turingův stroj je v podstatě velmi jednoduchý model počítače. Jedno z možných znázornění TS můžeme vidět na obrázku 2.1. Součástí Turingova stroje je:

- *páska* rozdělená na políčka. V každém políčku se nachází právě jeden symbol (popř. speciální prázdný symbol – tzv. blank), který je vybrán z určité (dopředu dané) konečné množiny symbolů. Symboly na jednotlivých políčkách pásky je možné nejen číst, ale i přepisovat. Páska je zleva ohraničená a směrem vpravo ji lze neomezeně prodlužovat o další políčka. Pásku si můžeme představit jako „paměť“ Turingova stroje.
- *čtecí/zápisová hlava*, která se může pohybovat po pásce směrem vlevo i vpravo a umožňuje čtení a přepis symbolů na jednotlivých políčkách pásky.
- *konečné stavové řízení*, které za pomocí čtecí/zápisové hlavy čte a provádí změny obsahu políček pásky. Můžeme si ho představit jako „program“ Turingova stroje.



Obrázek 2.1: Příklad znázornění Turingova stroje

Na začátku je na pásce TS zapsán vstupní řetězec (konečné délky) a to tak, že nejlevějším symbolem na pásce je speciální symbol blank a za ním následuje vlastní vstupní řetězec. Všechna políčka na pásce vpravo za vstupním řetězcem obsahují speciální prázdný symbol blank. Turingův stroj se v každém okamžiku výpočtu nachází vždy v právě jednom z jeho konečně mnoha stavů, který je uchován uvnitř stavového řízení stroje. Výpočet TS probíhá takto:

1. TS se na začátku výpočtu nachází v počátečním stavu a jeho čtecí/zápisová hlava je na nejlevější pozici pásky.

2. V každém kroku výpočtu Turingův stroj nejprve přečte aktuální symbol zapsaný na políčku pásky pod čtecí/zápisovou hlavou stroje, poté na základě tohoto symbolu a aktuálního stavu provede přechod, který zahrnuje následující akce:

   - Jedna z:
     - Zápis určitého symbolu na aktuální pozici na pásce (tím se přepíše symbol, který byl na začátku tohoto kroku čten).
     - Posun čtecí/zápisové hlavy stroje o jedno políčko pásky doprava.
     - Posun čtecí/zápisové hlavy stroje o jedno políčko pásky doleva.
   - Změna aktuálního stavu.

   Přechod, který bude v daném kroku výpočtu proveden, je určen tzv. přechodovou funkcí TS (viz definice 2.1). Tento bod je neustále opakován, možnosti ukončení výpočtu jsou popsány dále.

3. Možnosti, kdy dojde k ukončení výpočtu TS (provádění bodu 2), jsou pouze dvě:

   - První případ, kdy dojde k ukončení výpočtu je, pokud není pro aktuálně čtený symbol a aktuální stav definován žádný přechod. V tomto případě TS zastaví a:
     - Přijme svůj vstup, pokud se nachází ve speciálním koncovém stavu (normálně zastaví).
     - Jinak svůj vstup odmítne (abnormálně zastaví).

- Druhou možností je, že hlava TS je na nejlevější pozici pásky a dojde k posuvu hlavy doleva – říkáme, že Turingově stroji „přepadla“ hlava vlevo. V tomto případě stroj svůj vstup odmítne (abnormálně zastaví).

Může se ale také stát, že některé TS na některých vstupech budou běžet nekonečně dlouho (a tedy daný vstup ani nepřijmou ani neodmítnou). V tomto případě říkáme, že Turingův stroj na daném vstupu cyklí.

2.4 Formální definice Turingova stroje

Definice 2.1. Formálně je deterministický jednopáskový Turingův stroj definován jako šestice $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, kde:

- Q je konečná *množina vnitřních stavů* stroje.
- Σ je konečná množina symbolů nazývaná *vstupní abeceda*, $\Delta \notin \Sigma$.
- Γ je konečná množina symbolů nazývaná *pásková abeceda*, $\Sigma \subset \Gamma$, $\Delta \in \Gamma$.
- parciální funkce $\delta : (Q \setminus \{q_F\}) \times \Gamma \to Q \times (\Gamma \cup \{L, R\})$, kde $L, R \notin \Gamma$, je *přechodová funkce*.
- $q_0 \in Q$ je *počáteční stav*.
- $q_F \in Q$ je *koncový stav*.

Tolik k ryze formálnímu popisu TS, nyní si jeho jednotlivé části popišme detailněji:

- Q obsahuje stavy, ve kterých se může TS za běhu nacházet. Patří sem i dva speciální stavy: q_0 a q_F.
- Σ obsahuje symboly, které mohou být zapsány na pásce jako vstup Turingova stroje.
- Γ obsahuje symboly, které mohou být zapsány na pásce. Patří sem i speciální symbol Δ (tzv. blank). Jde o symbol představující prázdné políčko pásky a vyskytuje se tedy na všech jejích políčkách, na které nebylo dosud nic zapsáno (a protože platí $\Delta \in \Gamma$, může být Δ na pásku zapsán i v průběhu výpočtu TS). Symboly z $\Gamma \setminus \Sigma$ se mohou vyskytnout na pásce, ale nemohou být součástí vstupního řetězce TS – na pásce se tedy mohly vyskytnout pouze tak, že je tam TS zapsal (s výjimkou symbolu Δ).
- δ je parciální funkce, na základě které TS zjišťuje, jakou další akci má provést (můžeme si ji představit jako „program“ TS). Podívejme se blíže na její vstupy a výstupy:
  - Jako vstup funkce figuruje dvojice, ve které je první položkou stav TS s výjimkou koncového stavu q_F, což zajistí, že jakmile TS dospěje do q_F, tak ukončí výpočet a nebude pokračovat dále (narozdíl od konečných i zásobníkových automatů, které mohou provádět přechody i z koncových stavů). Druhou položkou vstupu funkce δ je aktuálně čtený symbol na pásce (což je symbol na pásce, který se nachází pod čtecí/zápisovou hlavou TS).

– Výstupem této funkce je potom opět dvojice: nový stav a buď jeden ze symbolů páskové abecedy nebo jeden ze speciálních symbolů L nebo R. Pokud je druhou položkou výstupu symbol páskové abecedy, je tento symbol zapsán na pásku na pozici, kde je aktuálně umístěna hlava TS. Pokud se jedná o speciální symbol L (resp. R) dojde k posuvu hlavy TS o jedno políčko pásky směrem doleva (resp. doprava).

Příklad 2.1. *Mějme TS M. Zápis $\delta(q, a) = (p, b)$, kde $q \in (Q \setminus \{q_F\})$, $p \in Q$, $a, b \in \Gamma$, znamená, že pokud se bude TS M s takto definovanou přechodovou funkcí nacházet ve stavu q a na pásce bude číst symbol a, má v dalším kroku výpočtu na pásku zapsat symbol b a přejít do stavu p.*

- q_0 je první ze dvou stavů se speciálním významem. Z tohoto stavu TS začíná provádět výpočet.
- q_F je druhým stavem se speciálním významem. Jak už bylo uvedeno výše, z tohoto stavu nemohou vést žádné přechody (nemůže figurovat jako vstupní stav do přechodové funkce δ) a pokud TS do tohoto stavu dospěje, úspěšně ukončí výpočet.

Pokud bychom chtěli dát uvedenou formální definici do vztahu k dříve zmíněnému neformálnímu popisu, můžeme si všimnout, že:

- Páska je zastoupena v Σ, Γ a částečně také v přechodové funkci δ.
- Čtecí/zápisová hlava je zastoupena pouze v přechodové funkci δ (viz popis výše).
- Konečné stavové řízení pak zahrnuje Q, δ, q_0 a q_F.

2.4.1 Konfigurace Turingova stroje

Definice 2.2. Konfigurace Turingova stroje M je definována jako trojice (q, z, n), kde:

- $q \in Q$ je aktuální stav TS.
- $z \in \{y\Delta^\omega \mid y \in \Gamma^*\}$ je nekonečný řetězec reprezentující aktuální obsah pásky TS (Δ^ω označuje nekonečný řetězec symbolů Δ).
- $n \in \mathbb{N}_0$ udává aktuální pozici čtecí/zápisové hlavy TS na pásce (0 značí nejlevější pozici pásky).

Z toho plyne, že konfigurace TS M je prvek množiny $Q \times \{y\Delta^\omega \mid y \in \Gamma^*\} \times \mathbb{N}_0$.

Konfigurace TS poskytuje všechny důležité informace o stavu výpočtu prováděném daným TS v určitém čase. Obvykle budeme konfigurace TS značit symboly $K_0, K_1, \ldots$.

Příklad 2.2. *Pro lepší pochopení výše uvedené definice si uveďme význačné příklady konfigurací TS M:*

- *Konfigurace $(q_0, \Delta w \Delta^\omega, 0)$, kde q_0 je počáteční stav a $w \in \Sigma^*$ je vstupní řetězec, se nazývá počáteční konfigurace. Z této konfigurace se standardně začíná provádět výpočet TS.*
- *Konfigurace (q_F, z, n), kde q_F je koncový stav, $z \in \Gamma^\omega$[1] a $n \in \mathbb{N}_0$, se nazývá koncová konfigurace.*

[1] Protože Γ je množina symbolů, Γ^ω značí množinu všech nekonečných řetězců nad Γ.

2.4.2 Přechodová relace a výpočet Turingova stroje

Nechť $z \in \Gamma^\omega$ a $n \in \mathbb{N}_0$, potom jako z_n označme n-tý symbol řetězce z (z_0 je nejlevější symbol). Dále nechť $a \in \Gamma$ a jako $s_a^n(z)$ označme řetězec, který vznikne z řetězce z záměnou (substitucí) n-tého symbolu (z_n) za symbol a.

Definice 2.3. Krok výpočtu TS M definujeme jako nejmenší binární relaci $\underset{M}{\vdash}$ takovou, že $\forall q_1, q_2 \in Q\ \forall z \in \Gamma^\omega\ \forall n \in \mathbb{N}_0\ \forall a \in \Gamma$:

- $(q_1, z, n) \underset{M}{\vdash} (q_2, z, n+1)$ pro $\delta(q_1, z_n) = (q_2, R)$, což odpovídá operaci posuvu doprava při z_n pod hlavou.
- $(q_1, z, n) \underset{M}{\vdash} (q_2, z, n-1)$ pro $\delta(q_1, z_n) = (q_2, L)$ a $n > 0$, což odpovídá operaci posuvu doleva při z_n pod hlavou.
- $(q_1, z, n) \underset{M}{\vdash} (q_2, s_a^n(z), n)$ pro $\delta(q_1, z_n) = (q_2, a)$, což odpovídá operaci zápisu a při z_n pod hlavou.

Příklad 2.3. *Přechod naznačený v příkladu 2.1 by potom mohl vypadat následovně:* $(q, z, n) \vdash (p, s_b^n(z), n)$, *kde* $z_n = a$.

Definice 2.4. Mějme TS M. Reflexivní a tranzitivní uzávěr $\underset{M}{\vdash}^*$ relace $\underset{M}{\vdash}$ je definován indukčně takto:

- $K_0 \underset{M}{\vdash}^0 K_0$, kde K_0 je konfigurace TS M.
- $K_0 \underset{M}{\vdash}^{n+1} K_2$ pokud $K_0 \underset{M}{\vdash}^n K_1 \underset{M}{\vdash} K_2$, kde K_0, K_1, K_2 jsou konfigurace TS M.
- $K_0 \underset{M}{\vdash}^* K_1$ pokud $K_0 \underset{M}{\vdash}^n K_1$ pro nějaké $n \in \mathbb{N}_0$.

Definice 2.5. Výpočet TS M na vstupním řetězci $w \in \Sigma^*$ začínající z konfigurace K_0 je posloupnost konfigurací $K_0, K_1, K_2, \ldots$, ve které platí $K_i \underset{M}{\vdash} K_{i+1}$ pro všechna $i \geq 0$ taková, že K_{i+1} je v dané posloupnosti. Tato posloupnost může být:

- nekonečná
- konečná s koncovou konfigurací (q, z, n), přičemž rozlišujeme následující typy zastavení TS:
  - normální – tzn. přechodem do koncového stavu (poté platí $q = q_F$)
  - abnormální:
    - $\delta(q, z_n)$ není definována
    - hlava TS je na nejlevější pozici pásky a dojde k posuvu hlavy doleva, tj. $\delta(q, z_n) = (q', L)$ pro nějaké $q' \in Q$

2.5 Jazyk přijímaný Turingovým strojem

Definice 2.6. Řekneme, že TS $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ přijal vstupní řetězec $w \in \Sigma^*$, jestliže $(q_0, \Delta w \Delta^\omega, 0) \underset{M}{\vdash}^* (q_F, z, n)$ pro nějaké $z \in \Gamma^\omega$ a $n \in \mathbb{N}_0$.

Pokud bychom si měli výše uvedenou definici popsat neformálně, tak řekneme, že TS M přijal vstupní řetězec w, jestliže M při aktivaci z počáteční konfigurace zastaví přechodem do koncového stavu.

Definice 2.7. Jazyk přijímaný Turingovým strojem M definujeme jako množinu $L(M) = \{w \mid w \in \Sigma^* \wedge w \text{ je přijat TS } M\}$.

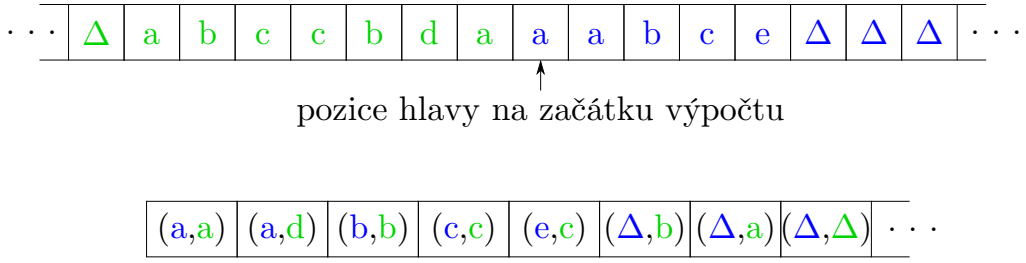
Kapitola 3

Varianty Turingových strojů

V této kapitole se budeme zabývat variantami Turingových strojů, které jsou definovány pomocí modifikací základní definice TS (definice 2.1) a které ale mají ekvivalentní výpočetní sílu. U každé varianty TS je vždy minimálně uveden její neformální popis a u vybraných variant je navíc přiblížena jejich formální definice a idea důkazu ekvivalence se základní podobou TS. V této kapitole jsem čerpal z [2, 5, 6, 8].

3.1 Turingův stroj s obousměrně nekonečnou páskou

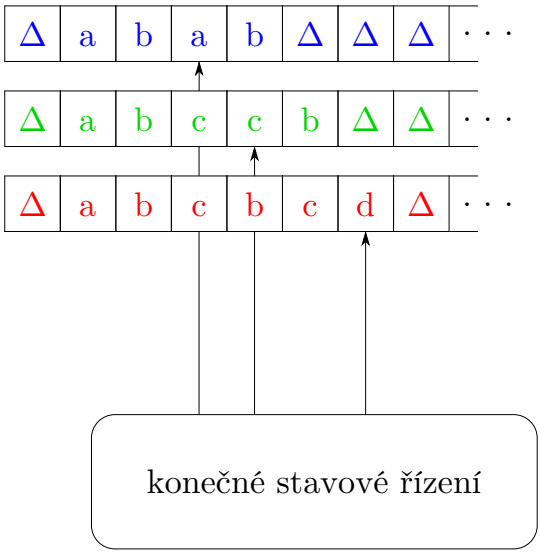
Turingův stroj, tak jak jsme si jej definovali, disponoval páskou, která byla zleva ohraničená a směrem vpravo nekonečná. Varianta TS, kterou se v této kapitole budeme zabývat, obsahuje pásku, která je nekonečná na obě strany ([6]). Takto „rozšířený“ TS je ale co se týče výpočetní síly shodný s TS z dřívější definice, protože TS s obousměrně nekonečnou páskou (označme jej TS M) lze základním modelem TS (označme jej TS N) simulovat. Konkrétně tak, že původní pásku „překlopíme“ doprava v bodě, kde je umístěna hlava na začátku výpočtu TS M (mimo toto políčko). Vznikne tak jednosměrně nekonečná páska, která má na každém svém políčku uloženy dva symboly. Tento problém lze vyřešit tak, že pásková abeceda TS N bude složena z dvojic (jelikož je počet symbolů v páskové abecedě TS M konečný, bude počet takto vytvořených dvojic také konečný). Poté už jen stačí přijmout konvenci, že např. první složky dvojic obsahují symboly umístěné vpravo od bodu „překlopení“ pásky a druhé složky obsahují symboly umístěné vlevo (v opačném pořadí). Pro lepší představu uvedené operace viz obrázek 3.1.



Obrázek 3.1: Simulace Turingova stroje s obousměrně nekonečnou páskou pomocí TS s pouze jednosměrně nekonečnou páskou

3.2 Vícepáskové Turingovy stroje

Jak už napovídá název, tyto varianty Turingova stroje obsahují více než jednu pásku. Obecně mohou obsahovat k pásek (kde $k \in \mathbb{N}$) – takový TS pak nazýváme k-páskový Turingův stroj ([5]). Pro daný k-páskový TS je k pevně dané a nelze ho měnit v průběhu výpočtu. k-páskový TS je složen z konečného stavového řízení (s téměř stejnou funkčností jako u jednopáskového TS), k pásek a stejného počtu čtecích/zápisových hlav, které jsou umístěny nad jednotlivými páskami stroje. Příklad třípáskového Turingova stroje M_1 je na obrázku 3.2 ([8]). Na začátku každého výpočtu je vstupní řetězec $w \in \Sigma^*$ zapsán stan-



Obrázek 3.2: Příklad vícepáskového Turingova stroje M_1

dardním způsobem na první pásce a všechny ostatní pásky jsou prázdné (na všech jejich políčkách je zapsán symbol blank; páskové abecedy jednotlivých pásek TS ale obecně nemusí být shodné). Jeden krok výpočtu k-páskového TS probíhá takto:

- TS nejprve přečte symboly pod každou hlavou na každé pásce.
- Na základě všech těchto symbolů a aktuálního stavu TS rozhodne, jakou další akci na svých páskách provede (patří sem stejně jako u jednopáskového TS zápis symbolu na aktuální pozici hlavy, posun hlavy na pásce vlevo nebo vpravo s tím, že tato akce je provedena pouze na jedné z k pásek tohoto stroje[1]) a následně přejde do nového stavu (vše podle přechodové funkce δ).

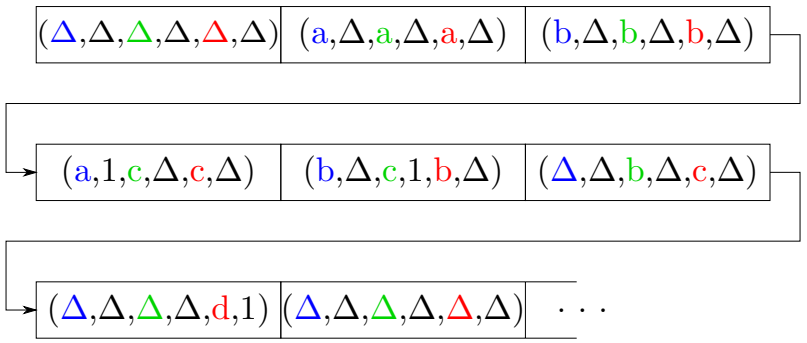
Pokud bychom chtěli k-páskový TS definovat formálně, bude jeho definice shodná s definicí jednopáskového TS s tím rozdílem, že bude navíc obsahovat definice páskových abeced všech pásek $(\Gamma_1, \Gamma_2, \ldots, \Gamma_k)$ a přechodová funkce bude mít tvar

$$\delta : (Q \setminus \{q_F\}) \times \Gamma_1 \times \Gamma_2 \times \ldots \times \Gamma_k \to Q \times \bigcup_{i \in \{1,\ldots,k\}} \{i\} \times (\Gamma_i \cup \{L, R\}),$$

[1] Existují ale také např. definice vícepáskových TS, které zmíněné akce provádějí na všech páskách, přičemž samozřejmě mohou provést na každé pásce jinou akci.

kde i značí pásku, na které má být daná akce provedena (zápis symbolu z odpovídající páskové abecedy nebo posuv hlavy).

Přestože se na první pohled může zdát, že vícepáskové Turingovy stroje mají větší výpočetní sílu než jednopáskové TS, ve skutečnosti tomu tak není. Mějme k-páskový TS M, ukážeme, jak k němu lze sestrojit ekvivalentní jednopáskový TS N (TS N bude TS M simulovat a tedy bude platit $L(M) = L(N)$). TS N bude mít páskovou abecedu speciálně rozšířenou tak, abychom na jeho jediné pásce mohli simulovat všech k pásek TS M. Konkrétně bude pásková abeceda TS N sestávat z $2k$-tic, kde vždy i-tá složka pro lichá i bude reprezentovat obsah $(\frac{i+1}{2})$-ní pásky TS M. Pro sudá i pak zde bude uložen jeden ze dvou symbolů: 1 nebo Δ podle toho, zda se na tomto políčku pásky (jejíž obsah je uložen na pozici $i-1$) nachází hlava TS M či nikoliv (pro ilustraci uvedené konstrukce je na obrázku 3.3 zachycen obsah pásky jednopáskového stroje TS N_1 odpovídající obsahům pásek třípáskového TS M_1 z obrázku 3.2). Protože jsou všechny páskové abecedy původního TS M konečné a na sudých pozicích v $2k$-tici se může vyskytnout pouze jeden ze dvou symbolů, je takto nově vytvořená rozšířená pásková abeceda TS N také konečná. TS N na začátku



Obrázek 3.3: Obsah pásky TS N_1 odpovídající obsahům pásek TS M_1 z obrázku 3.2

simulace nejprve převede obsah své pásky do formy $2k$-tic a poté už provádí vlastní simulaci kroků TS M. Při simulaci se využívá stavů ve formě $(k+1)$-tic, kde první složkou je stav původního TS M a zbylé složky jsou aktuálně čtené symboly na jednotlivých (simulovaných) páskách. Symboly uložené ve stavu stroje pak slouží při rozhodování o dalším kroku výpočtu k tomu, že TS N už nemusí pokaždé složitě zjišťovat aktuálně čtené symboly na jednotlivých páskách TS M. Po rozhodnutí o dalším kroku přesune TS N svou hlavu na pozici, na které je potřeba provést změnu – může se jednat o zápis symbolu na některou ze simulovaných pásek (odpovídá zápisu nové $2k$-tice na danou pozici) nebo o posuv některé ze simulovaných hlav (opět odpovídá zápisu nové $2k$-tice, ale tentokrát se musí modifikovat i $2k$-tice na pozici, která odpovídá políčku pásky, na které se simulovaná hlava přesunula). Za nový aktuální stav považujeme $(k+1)$-tici, která jako první složku obsahuje nový stav simulovaného stroje a zbylé složky zůstávají shodné se složkami z původního stavu TS N s výjimkou pozice, která odpovídá pásce, na které byly právě prováděny změny (obsah této složky stavu je patřičně aktualizován na aktuálně čtený symbol na dané pásce). TS N musí také korektně simulovat přepadnutí kterékoliv hlavy TS M na odpovídající pásce a také musí provádět převod dosud nevyužitých míst pásky obsahujících symbol blank na odpovídající $2k$-tici. Také musí platit, že TS N přejde do koncového stavu právě tehdy, když přejde do koncového stavu simulovaný TS M. Bez dalšího hlubšího popisu si uveďme, že takto zkonstruovaný TS N skutečně simuluje TS M a tedy platí $L(M) = L(N)$.

3.3 Nedeterministické Turingovy stroje

Doteď jsme vždy pracovali s deterministickými TS (dále také jako DTS), pro které platilo, že jejich přechodová funkce δ pro danou vstupní dvojici složenou z aktuálního stavu a čteného symbolu vracela *maximálně jednu* dvojici obsahující nový stav a buď jeden ze symbolů páskové abecedy nebo jeden ze speciálních symbolů L nebo R. Zmíněné „omezení“ ale už neplatí u nedeterministických TS (dále také jako NTS), u kterých přechodová funkce vrací obecně množinu takovýchto dvojic a NTS má tedy obecně několik možností pro následující krok výpočtu.

Formální definice NTS se opět oproti definici standardního DTS příliš neliší. Jediným rozdílem je, že přechodová funkce NTS je definována následovně:

- parciální funkce $\delta : (Q \setminus \{q_F\}) \times \Gamma \to 2^{Q \times (\Gamma \cup \{L,R\})}$, kde $L, R \notin \Gamma$, je *přechodová funkce*[2].

Jazyk přijímaný NTS je pak definován následovně.

Definice 3.1. Jazyk $L(M)$ NTS $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ je množina řetězců $w \in \Sigma^*$ takových, že M při aktivaci z počáteční konfigurace *může* zastavit přechodem do koncového stavu q_F.

Jinak řečeno NTS akceptuje vstupní řetězec právě tehdy, když existuje nějaká posloupnost výběru kroků vedoucí z počáteční do koncové konfigurace.

Nyní stejně jako u předchozí varianty TS přejdeme k nástinu důkazu, že NTS a DTS mají ekvivalentní výpočetní sílu. Mějme tedy nedeterministický TS M, ukážeme, jak k němu lze sestrojit ekvivalentní deterministický TS N (DTS N bude NTS M simulovat a tedy bude platit $L(M) = L(N)$). Simulace bude založena na tom, že DTS N bude postupně prohledávat všechny možné posloupnosti přechodů NTS M a bude zkoumat, zda některá z nich nevede do koncové konfigurace. DTS N bude konkrétně třípáskový TS[3] s následujícími významy jednotlivých pásek:

- 1. páska obsahuje přijímaný vstupní řetězec w. Obsah této pásky se v průběhu simulace nemění, slouží pouze pro obnovení počátečního obsahu pásky simulovaného stroje, pokud simulace vybrané posloupnosti přechodů NTS M nevedla do koncové konfigurace.
- 2. páska je pracovní páska. Je využívána pro simulaci pásky NTS M.
- 3. páska obsahuje v daném okamžiku vždy jeden řetězec, který v sobě kóduje jednu z možných posloupností přechodů NTS M (posloupnost přechodů je kódována posloupností čísel přiřazených přechodům simulovaného NTS M). Obsah této pásky je generován systematicky počínaje posloupností délky 0[4].

Vlastní simulace NTS M strojem N poté probíhá následovně:

1. Okopírujeme obsah 1. pásky na 2. pásku.

[2] Pro danou množinu A definujeme její potenční množinu jako množinu všech podmnožin dané množiny A a značíme ji 2^A.

[3] Jehož výpočetní síla je, jak jsme si uvedli v kapitole 3.2, ekvivalentní s jednopáskovým DTS.

[4] Pokud by NTS M např. obsahoval pouze tři přechody, které bychom kódovali jako desítkové číslice 1 až 3, obsahovala by tato páska při simulaci postupně řetězce ϵ, 1, 2, 3, 11, 12,....

2. Na 3. pásce generujeme příští posloupnost přechodů.

3. Simulujeme provedení posloupnosti přechodů z 3. pásky na obsahu 2. pásky.

4. Pokud vede zkoumaná posloupnost do koncového stavu q_F simulovaného stroje, DTS N zastaví a přijme. V opačném případě smažeme 2. pásku a vrátíme se do bodu 1.

Z uvedeného popisu je zřejmé, že jazyk přijímaný takto zkonstruovaným strojem je shodný s jazykem přijímaným původním nedeterministickým TS M, neboť:

- Pokud NTS M daný vstupní řetězec w přijme, pak existuje nějaká posloupnost přechodů, která vede ze stavu q_0 do stavu q_F stroje M. Protože DTS N provádí prohledávání množiny posloupností přechodů stroje M od nejkratších a postupně je prodlužuje, je jisté, že časem vygeneruje na třetí pásce i tuto přijímající posloupnost přechodů a tedy stroj N řetězec w také přijme.

- Naopak, pokud NTS M daný vstupní řetězec w nepřijme, pak neexistuje žádná posloupnost přechodů, která vede ze stavu q_0 do stavu q_F stroje M (jinak by z definice 3.1 byl w strojem M přijat). Pak ale ani DTS N během prohledávání množiny posloupností přechodů stroje M nemůže vygenerovat žádnou přijímající posloupnost přechodů a tedy stroj N řetězec w také nepřijme (konkrétněji nikdy nezastaví a bude množinu posloupností přechodů prohledávat do nekonečna).

3.4 Turingův stroj s oddělenou vstupní páskou

Vše, co si o této variantě Turingova stroje řekneme, je už vlastně součástí jeho názvu v nadpise podkapitoly. Jedná se o TS, který disponuje dvěma páskami: vstupní a pracovní. Na vstupní pásce je zapsán vstupní řetězec a z této pásky může stroj pouze číst (nesmí tedy měnit její obsah). Pracovní páska má pak stejnou funkcionalitu jako páska u klasického TS s tím, že pracovní páska na počátku obsahuje pouze symboly Δ.

3.5 Stroj se dvěma zásobníky

Poslední variantou Turingova stroje, kterou se zde budeme zabývat, je Stroj se dvěma zásobníky. Na tento stroj je možné nahlížet dvěma různými způsoby, které jsou ale ekvivalentní:

- Jedná se o Turingův stroj s oddělenou vstupní páskou, který má místo pracovní pásky dva zásobníky.

- Jde o deterministický zásobníkový automat se dvěma zásobníky.

Jak vyplývá z uvedeného popisu, stroj se dvěma zásobníky má tedy vstupní řetězec vložen na svém vstupu a nikoliv na pásce (zde zásobníku), jak tomu bylo zvykem u předchozích TS ([2]). Jeden krok výpočtu tohoto stroje pak závisí na:

- Aktuálním stavu konečného stavového řízení (možných stavů je konečný počet).

- Aktuálně čteném symbolu ze vstupu (symboly patří do konečné vstupní abecedy). Alternativně je povolen i ϵ-přechod (ze vstupu se nepřečte žádný symbol), ale musí být splněny podmínky determinismu.

- Symbolech umístěných na vrcholech obou zásobníků (symboly patří do konečné abecedy zásobníkových symbolů).

Vlastní provedení kroku výpočtu pak zahrnuje:

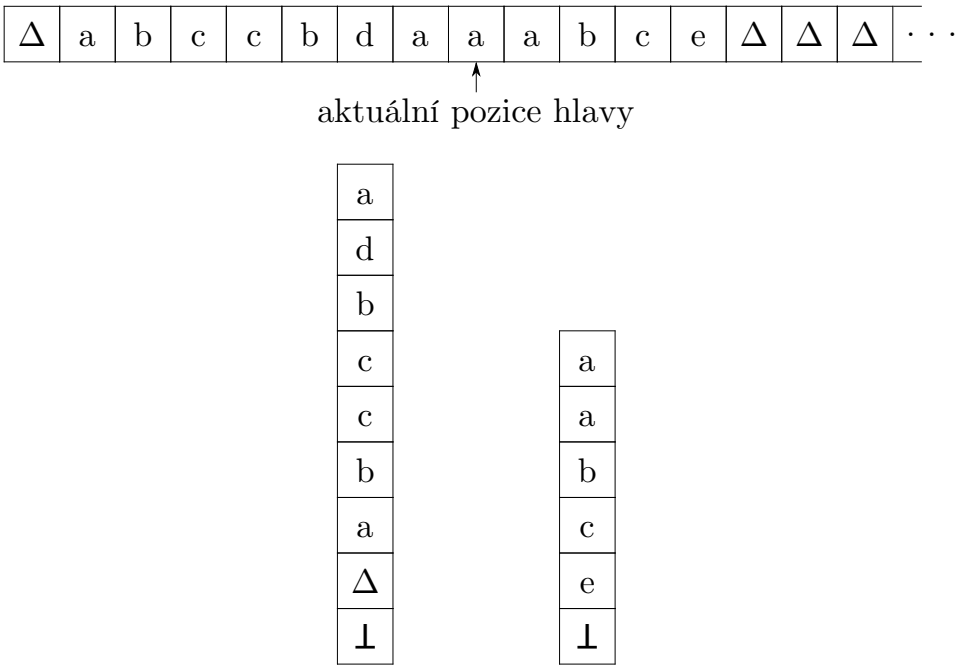
- Přechod do nového stavu.
- Nahrazení symbolů na vrcholech obou zásobníků řetězci o nula a více zásobníkových symbolech.

Co se týče výpočetní síly, je stroj se dvěma zásobníky ekvivalentní s jednopáskovým DTS. Zde si uvedeme pouze velmi zjednodušenou ideu důkazu tohoto tvrzení, detailní důkaz lze nalézt v [2].

Jednopáskový DTS lze simulovat pomocí stroje se dvěma zásobníky takto:

- Páska DTS je rozdělena na dvě poloviny: část vlevo od políčka, nad kterým je právě hlava DTS (dále jen hraniční políčko) a část vpravo zahrnující i zmíněné hraniční políčko.
- Obsah první části pásky je uložen do prvního zásobníku tak, že na vrcholu zásobníku je symbol bezprostředně vlevo od hraničního políčka.
- Obsah druhé části pásky (mimo nekonečně dlouhý řetězec obsahující symboly blank, který je umístěn napravo od nejpravějšího neblankového symbolu) je uložen do druhého zásobníku tak, že obsah hraničního políčka (právě čtený symbol simulovaného DTS) je na vrcholu zásobníku.
- Poté lze pohyby hlavy DTS doleva (resp. doprava) simulovat přesuny symbolu na vrcholu prvního (resp. druhého) zásobníku na vrchol druhého (resp. prvního) zásobníku.

Pro lepší představu uvedeného rozdělení pásky na dva zásobníky viz obrázek 3.4.



Obrázek 3.4: Příklad obsahu pásky TS a její reprezentace pomocí dvou zásobníků (symbol $\perp$ značí dno zásobníku)

Kapitola 4

Formy popisu Turingových strojů

V této kapitole budou uvedeny nejpoužívanější formy popisu Turingových strojů. Na začátku si uvedeme možnosti popisu Turingových strojů pomocí jejich formální definice a jí ekvivalentní grafické reprezentace v podobě přechodového diagramu. Dále si představíme popis TS pomocí kompozitních diagramů, kde se navíc zaměříme na to, v čem se liší kompozitní diagramy standardních TS od diagramů popisujících některé varianty TS. Na závěr si nastíníme možnosti popisu TS pomocí slovního vyjádření. V této kapitole jsem čerpal z [5].

4.1 Turingovy stroje zapsané pomocí formální definice

Prvním způsobem, jak můžeme popsat daný Turingův stroj, je pomocí formální definice, jak jsme ji uvedli v kapitole 2.4. Turingův stroj tedy popíšeme uvedením formálních označení všech jeho složek a u těch, u kterých je to potřeba, uvedeme i jejich detailnější formální popis – u množin jde o výpis jejich prvků, u přechodové funkce jde o uvedení všech jejích možných vstupů a odpovídajících výstupů. Bez dalšího vysvětlování si nejprve uveďme příklad takového popisu.

Příklad 4.1. *Mějme jazyk* $L = \{0^n 1^n \mid n \geq 1\}$. *Sestrojíme TS* M *takový, že* $L = L(M)$. *V tomto příkladě si TS* M *popíšeme pouze pomocí formální definice, v příkladech u následujících kapitol bude stejný TS popsán i dalšími způsoby. TS* M *je formálně definován jako šestice* $M = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$, *kde:*

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_F\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \Sigma \cup \{\Delta, x\}$
- δ*:*

$$\begin{aligned} \delta(q_0, \Delta) &= (q_1, R) & \qquad \delta(q_3, x) &= (q_3, R) \\ \delta(q_1, 0) &= (q_1, R) & \qquad \delta(q_3, 1) &= (q_2, x) \\ \delta(q_1, 1) &= (q_2, x) & \qquad \delta(q_3, \Delta) &= (q_4, L) \\ \delta(q_2, x) &= (q_2, L) & \qquad \delta(q_4, x) &= (q_4, L) \\ \delta(q_2, 0) &= (q_3, x) & \qquad \delta(q_4, \Delta) &= (q_F, x) \end{aligned}$$

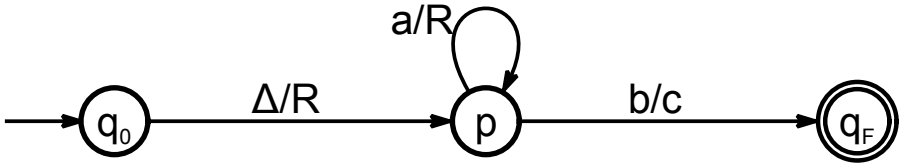
Pro vstupy, které zde nejsou uvedeny, není přechodová funkce δ *definována.*

V příkladu je na první pohled zřejmá asi největší nevýhoda formálního popisu TS – už pro jednoduché stroje je tento způsob popisu velmi nepřehledný a není z něj snadno čitelné, co vlastně popisovaný stroj dělá (příp. jak je definován jazyk, který tento stroj přijímá). Naopak jeho výhodou je, že nedává prostor pro nejasnosti (např. pokud bychom chtěli uvedený stroj simulovat), protože přesně definuje každý krok popisovaného stroje (což obecně nemusí platit např. u slovního popisu).

4.1.1 Přechodové diagramy Turingových strojů

Popis Turingových strojů pomocí přechodových diagramů patří do skupiny grafických reprezentací TS ([5]). Mimo jiné platí, že popis TS ve formě formální definice lze zcela mechanickým způsobem převést na popis ve formě přechodového diagramu. Naopak lze tuto konverzi provést také, ale protože v popisu pomocí přechodového diagramu nemusejí být některé části TS zachyceny celé, mohou příp. chybět také ve výsledné formální definici. Např. se v přechodovém diagramu nemusí objevit symbol vstupní abecedy, který sice dovolíme zapsat na vstup TS (formálně tedy patří do vstupní abecedy Σ daného TS), ale TS jej na pásku nikdy nezapíše a pokud jej TS na pásce nalezne, tak vstupní řetězec automaticky odmítne (což může být realizováno tak, že pro tento symbol nebude ze žádného stavu TS definován žádný přechod a tudíž se tento symbol nevyskytne v přechodovém diagramu takového TS).

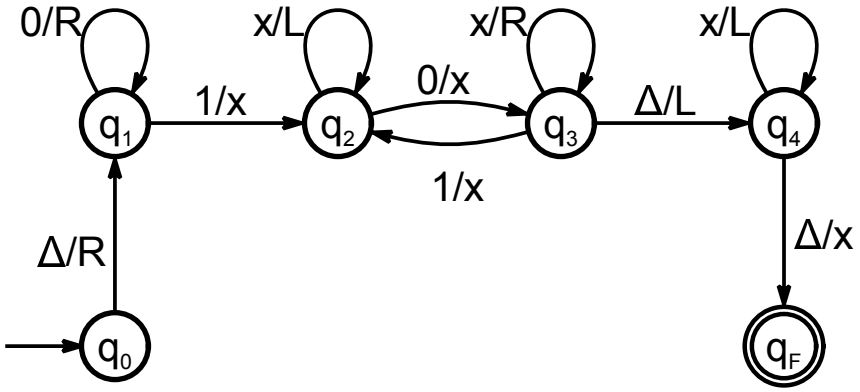
Nyní už přejdeme k vlastnímu popisu, jak se přechodové diagramy TS zakreslují. Vše si uvedeme na příkladu přechodového diagramu TS na obrázku 4.1. Stavy TS jsou v přecho-



Obrázek 4.1: Příklad přechodového diagramu jednoduchého Turingova stroje

dovém diagramu reprezentovány uzly, uvnitř kterých je uvedeno jejich označení. Sémantika hran v přechodovém diagramu je poté taková, že pokud v popisovaném Turingově stroji platí $\delta(q_1, a) = (q_2, x)$ (kde $q_1, q_2 \in Q, a \in \Gamma$ a x je buď symbol páskové abecedy Γ nebo jeden ze speciálních symbolů L nebo R), pak je tento fakt zachycen v přechodovém diagramu hranou vedoucí ze stavu q_1 do stavu q_2, která je popsaná jako a/x. Význam symbolů na hranách tedy je: „symbol, který se čte“ / „symbol, který se zapisuje“ (resp. symbol L nebo R pro posuv hlavy vlevo nebo vpravo). Počáteční stav je navíc označen hranou, která do tohoto stavu vstupuje, ale nemá stav, ze kterého by vycházela (na obrázku 4.1 jde o stav q_0). Koncový stav je pak reprezentován uzlem, který je zakreslen dvojicí soustředných kružnic (na obrázku 4.1 jde o stav q_F). Pro lepší ilustraci si na závěr ještě uveďme příklad přechodového diagramu TS M z příkladu 4.1.

Příklad 4.2. *Přechodový diagram TS M, pro který platí $L(M) = \{0^n 1^n \mid n \geq 1\}$, je na obrázku 4.2.*



Obrázek 4.2: Příklad přechodového diagramu TS M z příkladu 4.1

4.2 Kompozitní diagramy Turingových strojů

Další možností, jak můžeme Turingovy stroje popisovat, je pomocí kompozitních diagramů. Využívá se v nich faktu, že každý „složitý“ TS lze popsat kompozicí (spojením) „jednodušších“ TS. Nejprve si uvedeme konstrukce, jak lze TS spojovat do složitějších celků a poté jaké TS se běžně používají jako základní stavební bloky při konstrukci složitějších TS.

4.2.1 Konstrukce pro spojování TS do složitějších celků

Při konstrukci TS pomocí kompozitních diagramů předpokládáme, že všechny komponenty (Turingovy stroje, ze kterých výsledný TS skládáme) mají stejnou vstupní abecedu Σ i páskovou abecedu Γ. Konstrukce, které lze používat v kompozitních diagramech, budou nyní popsány blíže.

4.2.1.1 Bezpodmínečné předání řízení

Tato konstrukce se používá pro spojení Turingových strojů „za sebe“ do sekvence, tedy že nejprve je spuštěn výpočet jednoho TS a po jeho přechodu do koncového stavu je řízení předáno do počátečního stavu druhého TS. Graficky je tato konstrukce znázorněna orientovanou hranou mezi takto spojovanými komponentami. Např. pro TS A a B konstrukce A⟶B znamená předání řízení ze stavu q_F^A do stavu q_0^B, kde q_F^A je koncový stav stroje A a q_0^B je počáteční stav stroje B. Pokud bychom takto vzniklý výsledný TS (označme jej M_{AB}) chtěli popsat více formálně, pak M_{AB} zahrnuje všechny stavy a přechody obou komponent, stav q_F^A zde ztrácí status koncového stavu, stav q_0^B ztrácí status počátečního stavu a jsou doplněny přechody tak, že $\forall x \in \Gamma$: $\delta(q_F^A, x) = (q_0^B, x)$ (tedy pokud TS M_{AB} dojde do stavu q_F^A, tak beze změny obsahu pásky přejde do stavu q_0^B). Pokud má být výsledný TS popsaný kompozitním diagramem deterministický, musí platit, že z jedné komponenty může být řízení bezpodmínečně předáno do maximálně jedné komponenty. Uvedený zápis sekvence strojů A⟶B se často zkracuje na AB, je přitom dovoleno, aby příchozí hrany označující předání řízení směřovaly k libovolnému z TS v dané sekvenci. Řízení se potom předává tomu TS v dané sekvenci, ke kterému hrana směřuje (pro případ na obrázku 4.3a bude řízení předáno sekvenci AB, na obrázku 4.3b je řízení předáno již pouze stroji B). Stejným způsobem ale nelze pracovat s hranou znázorňující odchozí předání řízení – ta musí být spojena vždy s poslední komponentou v dané sekvenci.

(a) Předání řízení sekvenci AB

(b) Předání řízení stroji B

Obrázek 4.3: Bezpodmínečné předání řízení sekvenci strojů

4.2.1.2 Podmíněné předání řízení

Jak říká už samotný název, jde o podobnou konstrukci, jako je ta předchozí (předává se tedy řízení mezi určitými komponentami) s tím rozdílem, že předání řízení je podmíněno[1]. Podmínka předání řízení je, že se na políčku pásky pod hlavou stroje v daný okamžik (kdy se o předání řízení rozhoduje) vyskytne symbol z určité dané množiny symbolů (označme tuto množinu symbolů S; samozřejmě musí platit, že všechny symboly z množiny S patří do páskové abecedy Γ). Graficky je podmíněné předání řízení značeno stejně jako bezpodmínečné (tedy orientovanou hranou) s tím, že je u této orientované hrany navíc uvedena množina symbolů S. Narozdíl od bezpodmínečného předání řízení je možné, aby jedna komponenta měla i více odchozích podmíněných předání řízení, ovšem pouze za podmínky, že množiny symbolů vyskytující se u těchto podmíněných předání řízení jsou disjunktní. Sjednocení všech množin symbolů, které se vyskytují na odchozích hranách podmíněného předání řízení ale nemusí pokrývat celou páskovou abecedu Γ. Poté ovšem platí (na základě konvence zavedené v [5]), že pokud se pod čtecí hlavou vyskytne symbol, pro který není předání řízení explicitně stanoveno, výsledný TS automaticky přejde do koncového stavu, tedy **přijme**.

U podmíněného předání řízení je možné také vytvořit větev, která pokrývá všechny ostatní symboly, které nejsou pokryty uvedenými podmíněnými předáními řízení. Graficky se taková větev zakresluje stejně jako klasické podmíněné předání řízení s tím, že u hrany uvedeme množinu všech symbolů, přes které se už z dané komponenty podmíněně řízení předává, před kterou uvedeme symbol negace $\neg$. Jedná se vlastně jen o zjednodušený zápis. Ekvivalentně bychom mohli k hraně s takovou množinou symbolů uvést všechny symboly z páskové abecedy mimo ty ze zmíněné množiny (uvozené symbolem negace). Na závěr ještě uveďme, že kombinovat podmíněné a nepodmíněné předání řízení není u DTS možné.

Formálněji můžeme konstrukci podmíněného předání řízení popsat následovně. Předpokládejme, že z komponenty A máme $m \geq 1$ odchozích hran předávajících podmíněně řízení strojům $B_1, \ldots, B_m$ s množinami stavů $Q_1, \ldots, Q_m$, které jsou disjunktní navzájem a disjunktní s množinou stavů stroje A. Každý z těchto přechodů je označen symboly $x_1^i, \ldots, x_{n_i}^i \in \Gamma$, $n_i \geq 1$, $1 \leq i \leq m$. Výsledný TS zahrne všechny stavy a přechody původních strojů a navíc nový koncový stav q_F. Počáteční a koncové stavy q_0^i a q_F^i strojů B_i, $1 \leq i \leq m$, a koncový stav q_F^A stroje A ztrácejí svůj původní status. Navíc jsou přidány následující přechody:

- Pro všechna $1 \leq i \leq m$, $1 \leq j \leq n_i$, $y \in \Gamma \cup \{L, R\}$, $q \in Q_i$, je-li v B_i přechod $q_0^i \xrightarrow{x_j^i/y} q$, pak dodáme v kompozici přechod[2] $q_F^A \xrightarrow{x_j^i/y} q$.
- Pro každé $x \in \Gamma \setminus \{x_1^1, \ldots, x_{n_1}^1, \ldots, x_1^m, \ldots, x_{n_m}^m\}$ dodáme přechod $q_F^A \xrightarrow{x/x} q_F$.

[1] Dosáhneme tím na úrovni kompozitního diagramu možnosti větvit výpočet.

[2] Alternativně lze uvedené přechody nahradit následovně. Pro všechna $1 \leq i \leq m$, $1 \leq j \leq n_i$ dodáme v kompozici přechod $q_F^A \xrightarrow{x_j^i/x_j^i} q_0^i$.

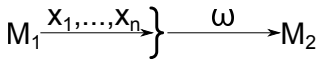
- Pro všechna $1 \leq i \leq m$ a $x \in \Gamma$ dodáme přechod $q_F^i \xrightarrow{x/x} q_F$.

Zjednodušeně řečeno jsou přidány následující přechody:

- Pokud k danému stroji B_i (vybranému ze strojů $B_1, \ldots, B_m$) vede v kompozitním diagramu ze stroje A hrana podmíněného předání řízení s určitým symbolem x a současně existuje v daném stroji B_i přechod z jeho počátečního stavu při čtení symbolu x na pásce do nějakého stavu q, pak je do výsledného TS přidán přechod z koncového stavu stroje A při čtení symbolu x na pásce do stavu q. Toto je provedeno pro všechny možné volby stroje B_i a symbolu x[3].
- Pro každý symbol, který se nevyskytuje v žádné z množin symbolů uvedených u hran podmíněného předání řízení vycházejících ze stroje A, je přidán přechod z koncového stavu stroje A při čtení tohoto symbolu na pásce do nově vytvořeného koncového stavu q_F.
- Pro každý symbol páskové abecedy a pro každý ze strojů $B_1, \ldots, B_m$ je dodán přechod z koncového stavu daného stroje při čtení daného symbolu na pásce do nově vytvořeného koncového stavu q_F.

4.2.1.3 Parametrová konvence

Zde se jedná o konstrukci, která je rozšířením konstrukce podmíněného předání řízení. Její grafický zápis do kompozitního diagramu je ukázán na obrázku 4.4. Množina symbolů v levé



Obrázek 4.4: Grafický zápis parametrové konvence v kompozitním diagramu

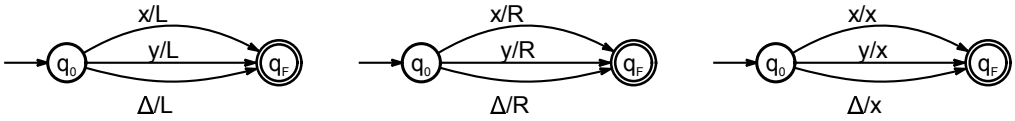
polovině grafického zápisu parametrové konvence má stejný význam jako u podmíněného předání řízení, tedy daný přechod je možné provést pouze za podmínky, že se na políčku pásky umístěném pod hlavou stroje aktuálně vyskytuje jeden ze symbolů uvedených v této množině ($x_1, \ldots, x_n \in \Gamma$, kde $n \geq 2$). Rozdílem oproti standardnímu podmíněnému předání řízení je pak to, že po provedení přechodu nabude symbol uvedený v pravé polovině grafického zápisu parametrové konvence ($\omega \notin \Gamma \cup \{L, R\}$) od tohoto okamžiku hodnoty symbolu, přes který byl tento přechod uskutečněn (aktuálně se vyskytuje na políčku pásky pod hlavou stroje). Tento speciální zástupný symbol pak lze používat v kompozitním diagramu standardním způsobem jako jiné symboly z páskové abecedy Γ.

4.2.2 Základní stavební bloky TS zapsaných kompozitními diagramy

V této kapitole pro jednoduchost uvažujme, že všechny uvedené Turingovy stroje mají shodnou páskovou abecedu $\Gamma = \{x, y, \Delta\}$. Mezi základní stavební bloky TS zapsaných pomocí kompozitních diagramů patří tyto stroje:

- Stroje L, R, x pro posuv hlavy stroje doleva, doprava, příp. pro zápis symbolu x na aktuální pozici na pásce (obrázek 4.5).

[3] Alternativně je možné namísto uvedených přechodů přidat následující. Pokud k danému stroji B_i vede v kompozitním diagramu ze stroje A hrana podmíněného předání řízení s určitým symbolem x, pak do výsledného TS přidáme přechod z koncového stavu stroje A beze změny obsahu pásky (či posuvu hlavy) do počátečního stavu stroje B_i. Toto je opět provedeno pro všechny možné volby stroje B_i a symbolu x.



(a) Stroj L (b) Stroj R (c) Stroj x

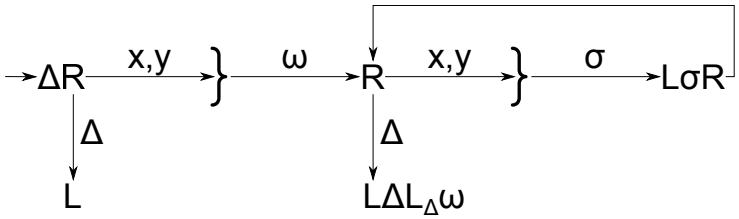
Obrázek 4.5: Základní stavební bloky TS

- Stroje $L_x, L_{\neg x}, R_x$ a $R_{\neg x}$ pro posuv hlavy stroje vlevo (resp. vpravo) na nejbližší symbol x příp. pro posuv vlevo (resp. vpravo) na nejbližší symbol různý od x (obrázek 4.6).

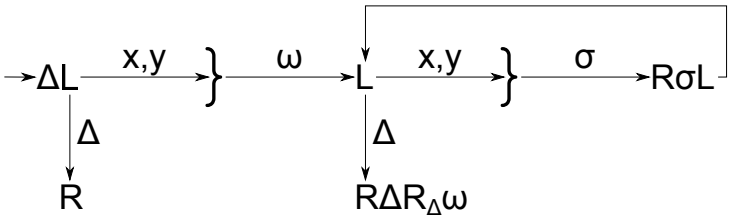
(a) Stroj L_x (b) Stroj $L_{\neg x}$ (c) Stroj R_x (d) Stroj $R_{\neg x}$

Obrázek 4.6: Stroje poskytující rozšířené možnosti posuvu hlavy stroje po pásce

- Stroje S_L a S_R pro posuv obsahu pásky vlevo resp. vpravo. Stroj S_L (resp. S_R) posune řetězec neblankových symbolů nacházejících se vpravo (resp. vlevo) od aktuální pozice hlavy o jeden symbol doleva (resp. doprava) (obrázek 4.7).



(a) Stroj S_L

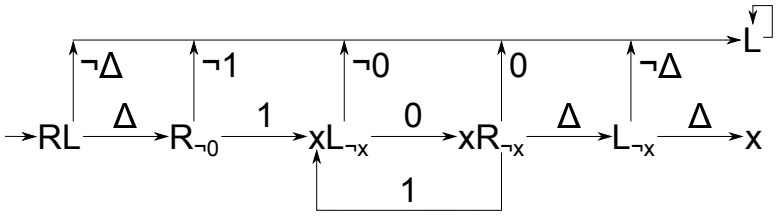


(b) Stroj S_R

Obrázek 4.7: Stroje pro posuv obsahu pásky

Na závěr podkapitoly si ještě uveďme příklad kompozitního diagramu nám dobře známého TS M z příkladu 4.1.

Příklad 4.3. *Kompozitní diagram TS M, pro který platí $L(M) = \{0^n1^n \mid n \geq 1\}$, je na obrázku 4.8.*



Obrázek 4.8: Příklad kompozitního diagramu TS M z příkladu 4.1

4.2.3 Poznámky k variantám Turingových strojů

V této kapitole si uvedeme rozdíly v zápisu kompozitních diagramů popisujících některé varianty Turingových strojů oproti výše popsané konstrukci těchto diagramů určených pro standardní podobu Turingova stroje.

4.2.3.1 Vícepáskové Turingovy stroje

K popisu vícepáskových TS pomocí kompozitních diagramů lze využít výše popsaných konstrukcí se stejným významem. Jediným rozdílem je, že je třeba v takovém diagramu u každého symbolu i základní stavební komponenty vždy uvést označení pásky, se kterou se v daném případě pracuje. Konkrétněji:

- U podmíněného předání řízení je třeba u každého symbolu zapsaného na hraně uvést označení pásky pomocí horního indexu (např. x^1, y^2), na které se má daný symbol nacházet, aby bylo možné toto předání řízení provést. Současně je ale také možné i podmíněné předání řízení na základě určitých kombinací symbolů čtených z různých pásek (zapisováno jako např. $x^1 \wedge y^2$, příp. lze použít i ekvivalentní zápis $x^1 y^2$). Orientovaná hrana označená např. $x^1, y^1 z^2$ pak znamená, že předání řízení se provede pouze tehdy, pokud je pod čtecí hlavou na první pásce symbol x, nebo pokud je pod čtecí hlavou na první pásce symbol y a současně je pod čtecí hlavou na druhé pásce symbol z.
- U základních stavebních komponent uvedených v předchozí podkapitole je v kompozitním diagramu vícepáskových TS opět potřeba pomocí horního indexu označit pásku, na které má daná komponenta pracovat (např. $\mathsf{L}^1\mathsf{R}^2$).

4.2.3.2 Nedeterministické Turingovy stroje

K popisu nedeterministických TS pomocí kompozitních diagramů lze opět využít konstrukcí popsaných v kapitolách 4.2.1 a 4.2.2 se stejným významem. U NTS je ale možné používat některé konstrukce i způsoby, které byly u DTS zakázány. Konkrétně je v kompozitních diagramech NTS navíc možné:

- Aby množiny symbolů uvedené u hran podmíněného předání řízení nebyly disjunktní.
- Aby množiny symbolů uvozené symbolem ¬ uvedené u hran podmíněného předání řízení netvořily svým významem přesný doplněk množiny symbolů vyskytujících se v množinách symbolů u ostatních hran podmíněného předání řízení (doplněk vzhledem k páskové abecedě).

- Aby bylo kombinováno podmíněné a nepodmíněné předání řízení. V tomto případě se ale nedoplňují implicitní přechody do nového koncového stavu q_F přes symboly, které nejsou pokryty žádným podmíněným předáním řízení.

4.3 Turingovy stroje popsané slovně (neformálně)

Poslední formou popisu, kterou si představíme, je slovní popis. Jde pouze o slovní vyjádření toho, co by měl popisovaný TS dělat v různých nastalých situacích. Uveďme si příklad takového popisu na TS M z příkladu 4.1.

Příklad 4.4. *Slovní popis TS M, pro který platí $L(M) = \{0^n 1^n \mid n \geq 1\}$, by mohl vypadat např. takto:*

1. *TS M startuje svůj výpočet se standardním obsahem pásky (tj. $\Delta w \Delta^\omega$, kde $w \in \Sigma^*$) a hlavu má umístěnou na nejlevějším políčku pásky.*

2. *V prvním kroku stroj ověří, že na prvním políčku pásky je skutečně symbol Δ. Pokud tomu tak není, abnormálně zastaví.*

3. *M posune hlavu doprava za symbol Δ. Následně M posouvá hlavu doprava tak dlouho, dokud se nedostane na políčko, na kterém je zapsán symbol různý od symbolu 0 (posune se tedy na první symbol nacházející se za řetězcem skládajícím se pouze ze symbolů 0). Pokud je tímto symbolem 1, M pokračuje dál (jinak abnormálně zastaví).*

4. *M zapíše na aktuální pozici symbol x a následně posouvá hlavu doleva, dokud nenarazí na symbol nacházející se před řetězcem skládajícím se ze symbolů x. Pokud je tímto symbolem 0, zapíše na tuto pozici symbol x (jinak abnormálně zastaví).*

5. *M posouvá hlavu doprava na první symbol nacházející se za řetězcem skládajícím se pouze ze symbolů x. Pokud je tímto symbolem:*

   - 1 – *přejde do bodu 4,*
   - Δ – *pokračuje dále,*
   - 0 – *abnormálně zastaví.*

6. *M posune hlavu doleva před symbol Δ. Následně M posouvá hlavu doleva, dokud nenarazí na symbol nacházející se před řetězcem skládajícím se ze symbolů x. Pokud je tímto symbolem Δ, zapíše na aktuální pozici symbol x a normálně zastaví (tj. přijme vstupní řetězec). Jinak zastaví abnormálně.*

Uvedený slovní popis TS M je na nízké úrovni abstrakce – popisujeme v něm takřka každý krok uvedeného stroje. Slovní popis má ale oproti jiným formám popisu zejména tu výhodu, že pomocí něj můžeme TS popisovat také na vysoké úrovni abstrakce z čehož mimo jiné vyplývá, že pomocí něj lze popisovat i velmi složité TS s tím, že takovýto popis je narozdíl od dříve uvedených technik pro člověka stále relativně srozumitelný. Uveďme si příklad takového popisu opět na TS M z příkladu 4.1.

Příklad 4.5. *TS M na začátku ověří, že na prvním políčku pásky je symbol Δ a následně přesune hlavu za řetězec složený ze symbolů 0 (čímž se v případě správně zadaného vstupu dostane těsně za střed vstupního řetězce). Nyní bude TS postupně od středu řetězce vyškrtávat střídavě symboly 1 a 0 (přepíše je např. pomocí symbolu x), dokud nenastane situace,*

kdy TS vyškrtl poslední pár symbolů 0 *a* 1*. V tom případě TS ještě zkontroluje, zda se za koncem nebo před začátkem „proškrtaného“ řetězce nevyskytují nějaké zbylé symboly* 0 *nebo* 1 *a pokud ne, normálně zastaví. Ve všech ostatních nepopsaných případech stroj abnormálně zastaví.*

Kapitola 5

Problém zastavení Turingova stroje

V této kapitole se budeme zabývat problémem zastavení Turingova stroje, což je v teoretické informatice důležitý problém, jehož použití bude vysvětleno dále. Ale ještě před tím, než přistoupíme k definici tohoto problému, bude na začátku potřeba zavést pojmy rekurzivně vyčíslitelných a rekurzivních jazyků, rozhodovacích problémů a jejich klasifikace, kódování Turingova stroje a také pojem univerzálního Turingova stroje. V této kapitole jsem čerpal z [5, 16].

5.1 Rekurzivně vyčíslitelné a rekurzivní jazyky

Před zavedením pojmů rekurzivně vyčíslitelných a rekurzivních jazyků si nejprve uveďme následující definici úplného Turingova stroje ([5]).

Definice 5.1. Turingův stroj se nazývá úplný právě tehdy, když zastaví pro každý vstup.

Definice 5.2. Jazyk $L \subseteq \Sigma^*$ se nazývá:

- *rekurzivně vyčíslitelný*, jestliže $L = L(M)$ pro nějaký TS M,
- *rekurzivní*, jestliže $L = L(M)$ pro nějaký úplný TS M.

Dále o vztahu mezi TS a uvedenými třídami jazyků platí následující:

- Řekneme, že Turingův stroj M rozhoduje jazyk $L(M)$, pokud je M úplný.
- Ke každému rekurzivnímu jazyku existuje TS, který ho rozhoduje, tedy zastaví pro každý vstup.
- Ke každému rekurzivně vyčíslitelnému jazyku L existuje TS, který zastaví pro každé $w \in L$. Pro $w \notin L$ může stroj zastavit, ale také může donekonečna cyklit.

5.2 Rozhodovací problémy a jejich klasifikace

Rozhodovací problém P (dále také jako problém P) obvykle chápeme jako funkci f_P s oborem hodnot {true, false}. Rozhodovací problém je standardně specifikován dvojicí množin (A_P, B_P), kde:

- A_P je definiční obor funkce f_P, tedy množina možných instancí problému (možných vstupů).

- $B_P \subseteq A_P$ je podmnožina instancí, pro které je hodnota funkce f_P rovna true ($B_P = \{p \mid f_P(p) = true\}$).

Pokud ke kódování jednotlivých instancí problémů použijeme řetězce nad vhodnou abecedou Σ, je pak možné rozhodovací problém P specifikovat jazykem $L_P = \{w \in \Sigma^* \mid w = code(p), p \in B_P\}$, kde $code : A_P \to \Sigma^*$ je injektivní funkce, která nezávisle na f_P přiřazuje instancím problému příslušný řetězec.

Na závěr této podkapitoly si uveďme důležitou klasifikaci rozhodovacích problémů na rozhodnutelné, nerozhodnutelné a částečně rozhodnutelné.

Definice 5.3. Nechť P je problém specifikovaný jazykem L_P nad Σ. Problém P nazveme:

- *Rozhodnutelný*, pokud L_P je rekurzivní jazyk, tj. existuje TS, který L_P rozhoduje, tedy přijme každý řetězec $w \in L_P$ a zamítne každý řetězec $w \in \Sigma^* \setminus L_P$.
- *Nerozhodnutelný*, když není rozhodnutelný.
- *Částečně rozhodnutelný*, jestliže L_P je rekurzivně vyčíslitelný jazyk, tj. existuje TS, který přijme každý řetězec $w \in L_P$ a každý řetězec $w \in \Sigma^* \setminus L_P$ zamítne, nebo je jeho výpočet na něm nekonečný.

Z uvedené definice plyne, že:

- Každý problém je buď rozhodnutelný a nebo nerozhodnutelný.
- Pro některé nerozhodnutelné problémy ale platí, že jsou částečně rozhodnutelné.
- Všechny rozhodnutelné problémy jsou zároveň částečně rozhodnutelné.

5.3 Kódování Turingova stroje

Kódováním TS máme na mysli jednoznačný popis TS ve formě řetězce. Motivací pro zakódování TS do řetězce je možnost zadat tento TS na vstup univerzálního TS (viz kapitola 5.4). Dále uvedený způsob kódování TS je převzat z [5], existují však i jiné přístupy (např. jako ty v [8, 17]).

Kódovací systém pro TS zahrnuje:

- Kódování stavů (všech stavů TS včetně q_0 a q_F).
- Kódování symbolů z páskové abecedy Γ.
- Kódování přechodové funkce δ.

Kódování stavů – Množinu stavů Q uspořádáme do posloupnosti $q_0, q_F, p, q, \ldots, t$. Stav, který se v posloupnosti nachází na j-té pozici (označme jej q_j), zakódujeme jako 0^j, přičemž indexujeme např. od nuly.

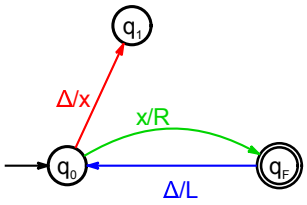
Kódování symbolů a příkazů L/R – Předpokládejme, že $\Gamma = \Sigma \cup \{\Delta\}$. Uspořádáme Σ do posloupnosti $a_1, a_2, \ldots, a_n$ a zvolíme tyto kódy:

$$\Delta \mapsto \epsilon \qquad L \mapsto 0 \qquad R \mapsto 00 \qquad a_i \mapsto 0^{i+2}$$

Kódování přechodové funkce δ – Přechod $\delta(p, x) = (q, y)$, kde $x \in \Gamma, y \in \Gamma \cup \{L, R\}$, reprezentujeme čtveřicí (p, x, q, y) a kódujeme zřetězením kódů p, x, q, y při použití symbolu 1 jako oddělovače. Výsledný kód má tedy tvar $\langle p\rangle 1\langle x\rangle 1\langle q\rangle 1\langle y\rangle$, kde $\langle r\rangle$ značí kód r.

Celý TS pak kódujeme jako posloupnost kódů přechodů oddělených a ohraničených symbolem 1.

Příklad 5.1. *Mějme TS M', jehož přechodový diagram je na obrázku 5.1. Kód TS M' je: 11100100011000101001011101.*



Obrázek 5.1: Přechodový diagram TS M'

5.4 Univerzální Turingův stroj

Univerzální TS je neformálně řečeno TS schopný simulovat výpočet jakéhokoliv TS M na libovolném vstupním řetězci w. Jde tedy o „programovatelný“ TS, jehož vstupem je M (můžeme chápat jako „program“) a w (můžeme chápat jako „data“) a který spustí daný program nad danými daty.

Aby bylo možné zapsat simulovaný TS M na pásku univerzálního TS, je třeba TS M vhodně zakódovat. Pro tento účel použijeme kódování zavedené v kapitole 5.3. Vstupní řetězec w budeme kódovat jako posloupnost příslušných kódů symbolů oddělených a ohraničených symbolem 1. Kód stroje M a vstupního řetězce w oddělíme např. symbolem #.

Příklad 5.2. *Pro ilustraci si uveďme příklad vstupu univerzálního TS. Pokud bychom chtěli pomocí univerzálního TS spustit TS M' z příkladu 5.1 na řetězci xx jako jeho vstupu, zapsali bychom na vstup univerzálního TS kód: 11100100011000101001011101#100010001.*

Nyní už můžeme přistoupit k popisu vlastního univerzálního TS. Způsobů, jak můžeme univerzální TS navrhnout, existuje více, my si představíme jeden z nich. Konkrétně se bude jednat o třípáskový TS, kde význam jednotlivých pásek je následující:

- 1. páska je využita pro uložení vstupu (v zakódované podobě jak bylo uvedeno výše) a později výstupu.
- 2. páska je používána k simulaci pracovní pásky původního stroje M.
- 3. páska slouží pro uložení aktuálního stavu simulovaného stroje M a aktuální pozice hlavy (pozice hlavy i je zakódována jako 0^i).

Univerzální TS pak pracuje takto:

1. Stroj zkontroluje, zda vstup odpovídá zadání. Pokud ne, abnormálně zastaví.
2. Stroj zkopíruje w na 2. pásku, na 3. pásku zapíše kód stavu q_0 a za něj poznačí, že hlava stroje M se nachází na nejlevějším políčku pásky.
3. Na 2. pásce stroj na základě aktuální pozice hlavy simulovaného stroje vyhledá aktuálně čtený symbol (označme jej x). Dále se na 1. pásce pokusí nalézt přechod proveditelný ze stavu zapsaného na 3. pásce pro symbol x. Pokud se žádný takový přechod nenalezne, stroj abnormálně zastaví.

4. Stroj provede na 2. a 3. pásce změny odpovídající simulovanému přechodu, tj. přepis aktuálního symbolu, změna pozice hlavy a změna aktuálního stavu simulovaného stroje M.

5. Pokud nebyl dosažen koncový stav q_F simulovaného stroje, přejdeme na bod 3. Jinak stroj vymaže 1. pásku, překopíruje na ni obsah 2. pásky a zastaví normálně (přechodem do **svého** koncového stavu q_F).

Uvedený stroj můžeme převést na jednopáskový univerzální TS, který budeme dále značit T_U.

5.5 Problém zastavení Turingova stroje

Následující věta a její důkaz byly převzaty z [5].

Věta 5.1. Problém zastavení TS (angl. Halting problem – *HP*), kdy nás zajímá, zda daný TS M pro daný vstupní řetězec w zastaví, není rozhodnutelný, ale je částečně rozhodnutelný.

Důkaz. Problému zastavení odpovídá rozhodování jazyka $HP = \{\langle M\rangle\#\langle w\rangle \mid M \text{ zastaví na vstupu } w\}$, kde $\langle M\rangle$ je kód TS M a $\langle w\rangle$ je kód w.

Částečnou rozhodnutelnost lze ukázat snadno použitím modifikovaného T_U, který **zastaví přijetím** vstupu $\langle M\rangle\#\langle w\rangle$ právě tehdy, když M **zastaví** (ne nutně přijetím) na vstupu w. Modifikace spočívá v převedení abnormálního zastavení při simulaci na zastavení přechodem do stavu q_F.

Nerozhodnutelnost ukážeme s využitím techniky diagonalizace:

- Pro $x \in \{0,1\}^*$, nechť M_x je TS s kódem x, je-li x legální kód TS. Jinak ztotožníme M_x s pevně zvoleným TS (např. s TS, který pro libovolný vstup okamžitě zastaví).
- Můžeme nyní sestavit posloupnost $M_\epsilon, M_0, M_1, M_{00}, M_{01}, M_{10}, M_{11}, M_{000}, \ldots$ zahrnující všechny TS nad $\Sigma = \{0,1\}$ indexované řetězci z $\{0,1\}^*$.
- Uvažme nekonečnou matici

	$\boldsymbol{\epsilon}$	**0**	**1**	**00**	**01**	**10**	...
$\boldsymbol{M_\epsilon}$	$H_{M_\epsilon,\epsilon}$	$H_{M_\epsilon,0}$	$H_{M_\epsilon,1}$	$H_{M_\epsilon,00}$	$H_{M_\epsilon,01}$	...	
$\boldsymbol{M_0}$	$H_{M_0,\epsilon}$	$H_{M_0,0}$	$H_{M_0,1}$	$H_{M_0,00}$	$H_{M_0,01}$	...	
$\boldsymbol{M_1}$	$H_{M_1,\epsilon}$	$H_{M_1,0}$	$H_{M_1,1}$	$H_{M_1,00}$	$H_{M_1,01}$	...	
$\boldsymbol{M_{00}}$	$H_{M_{00},\epsilon}$	$H_{M_{00},0}$	$H_{M_{00},1}$	$H_{M_{00},00}$	$H_{M_{00},01}$	...	
$\boldsymbol{M_{01}}$	$H_{M_{01},\epsilon}$	$H_{M_{01},0}$	$H_{M_{01},1}$	$H_{M_{01},00}$	$H_{M_{01},01}$	...	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	

  kde $H_{M_x,y} = \begin{cases} \text{C, jestliže } M_x \text{ cyklí na } y. \\ \text{Z, jestliže } M_x \text{ zastaví na } y. \end{cases}$
- Předpokládejme, že existuje úplný TS K přijímající jazyk *HP*. Pak K pro vstup $\langle M\rangle\#\langle w\rangle$:
  - Zastaví normálně (přijme) právě tehdy, když M zastaví na w.
  - Zastaví abnormálně (odmítne) právě tehdy, když M cyklí na w.

- Sestavíme TS N, který pro vstup $x \in \{0,1\}^*$:
  - Sestaví M_x z x a zapíše $\langle M_x \rangle \# x$ na svou pásku.
  - Simuluje K na $\langle M_x \rangle \# x$ a:
    - Přijme, pokud K odmítne.
    - Přejde do nekonečného cyklu, pokud K přijme.

  Všimněme si, že N v podstatě komplementuje diagonálu uvedené matice:

	$\boldsymbol{\epsilon}$	**0**	**1**	**00**	**01**	**10**	...
$\boldsymbol{M_\epsilon}$	$H_{M_\epsilon,\epsilon}$	$H_{M_\epsilon,0}$	$H_{M_\epsilon,1}$	$H_{M_\epsilon,00}$	$H_{M_\epsilon,01}$	...	
$\boldsymbol{M_0}$	$H_{M_0,\epsilon}$	$H_{M_0,0}$	$H_{M_0,1}$	$H_{M_0,00}$	$H_{M_0,01}$	...	
$\boldsymbol{M_1}$	$H_{M_1,\epsilon}$	$H_{M_1,0}$	$H_{M_1,1}$	$H_{M_1,00}$	$H_{M_1,01}$	...	
$\boldsymbol{M_{00}}$	$H_{M_{00},\epsilon}$	$H_{M_{00},0}$	$H_{M_{00},1}$	$H_{M_{00},00}$	$H_{M_{00},01}$	...	
$\boldsymbol{M_{01}}$	$H_{M_{01},\epsilon}$	$H_{M_{01},0}$	$H_{M_{01},1}$	$H_{M_{01},00}$	$H_{M_{01},01}$	...	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	

- Dostáváme, že:

$$N \text{ zastaví na } x \Leftrightarrow {}^{1}K \text{ odmítne } \langle M_x \rangle \# x \Leftrightarrow {}^{2}M_x \text{ cyklí na } x$$

- To ale znamená, že N se liší od každého M_x alespoň na jednom řetězci – konkrétně na x. To je ovšem spor s tím, že posloupnost $M_\epsilon, M_0, M_1, M_{00}, M_{01}, M_{10}, M_{11}, M_{000}, \ldots$ zahrnuje všechny TS nad $\Sigma = \{0,1\}$.

  Tento spor plyne z předpokladu, že existuje TS K, který pro daný TS M a daný vstup x určí (rozhodne), zda M zastaví na x či nikoliv.

□

Tímto jsme ukázali, že jazyk *HP* je rekurzivně vyčíslitelný, ale není rekurzivní. Z toho pak plyne, že problém zastavení TS je jen částečně rozhodnutelný. Zmíněné skutečnosti se často využívá v důkazech, že daný problém není rozhodnutelný, neboli že odpovídající jazyk není rekurzivní (konkrétně se používá technika redukce – více viz [5]).

5.6 Fundované uspořádání (angl. well-founded order)

Z předchozích kapitol víme, že problém zastavení je nerozhodnutelný. Přesto je v praxi často potřeba jej (alespoň pro některé případy) umět rozhodnout. K nejzákladnějším přístupům, pomocí kterých lze alespoň částečně problém zastavení řešit, patří přístupy založené na tzv. fundovaných uspořádáních. Dále bude uveden jednoduchý princip využívající fundovaných uspořádání pro rozhodování některých instancí problému zastavení Turingova stroje. Nejprve si ale uveďme definici fundovaného uspořádání ([16]).

Definice 5.4. Říkáme, že relace uspořádání $<$ je fundovaná, jestliže neobsahuje nekonečnou klesající posloupnost.

[1] Dle definice N.
[2] Předpoklad o K.

Jako příklad fundovaného uspořádání si uveďme relaci $<$ („je menší než“) definovanou na $\mathbb{N}_0$. Protipříkladem pak může být stejná relace definovaná na $\mathbb{Z}$.

S využitím fundovaného uspořádání pak můžeme pro rozhodování některých instancí problému zastavení TS použít následující princip. Uvažme, že v přechodovém diagramu vyšetřovaného TS existuje právě jeden cyklus. Potom můžeme s jistotou říci, že TS vždy zastaví, pokud bude platit, že výsledkem jednoho provedení cyklu bude taková změna v konfiguraci TS, že některá z položek nové konfigurace je „menší“ (podle předem definovaného fundovaného uspořádání) než odpovídající položka v původní konfiguraci (viz příklad dále). Abychom mohli u složitějšího TS říci, že vždy zastaví, pak musí tato podmínka platit pro všechny jeho cykly. Podobně lze uvedený princip rozšířit na TS zapsané pomocí kompozitních diagramů.

Příklad 5.3. *Předpokládejme, že jsme analýzou TS zjistili, že provedením jedné iterace určitého cyklu v přechodovém diagramu přejde TS z konfigurace* $K_1 = (q_1, \Delta w_1 \Delta^\omega, n_1)$ *do* $K_2 = (q_2, \Delta w_2 \Delta^\omega, n_2)$*, přičemž bude vždy platit* $n_2 < n_1$ *(*$<$ *je klasické uspořádání „je menší než“ na* $\mathbb{N}_0$*). Pak protože* $<$ *je fundované uspořádání, můžeme si být jisti, že provádění tohoto cyklu někdy v budoucnu skončí.*

Pro lepší pochopení předchozího příkladu si stačí uvědomit, že výsledkem provedení každé iterace uvedeného cyklu je posuv hlavy o nějaký počet políček pásky směrem vlevo. Je tedy jisté, že buď cyklus skončí sám (splněním nějaké definované podmínky), a nebo dříve či později dojde k pokusu o posuv hlavy vlevo z nejlevějšího políčka pásky a tedy k abnormálnímu zastavení Turingova stroje. Tento koncept lze dále rozšířit i na obsah pásek a tedy fundované uspořádání vytvářet nad symboly páskové abecedy TS. Zde pak sledujeme, jakým způsobem dochází k přepisování symbolů na pásce a snažíme se zjistit, zda přepisování nesplňuje určitá pravidla, která by nám zajistila, že někdy v budoucnu dojde k situaci, kdy TS už nebude mít další symboly na přepisování a tedy jeho výpočet skončí.

Kapitola 6

Složitost výpočtů Turingova stroje

V této kapitole se stručně podíváme na problém analýzy složitosti výpočtů Turingova stroje. Ještě předtím si ale uvedeme rozdíly mezi teoretickou a praktickou řešitelností problémů a následně také motivaci, proč se zabývat studiem analýzy složitosti výpočtů realizovaných na TS v podobě popisu problémů, které mohou nastat při analýze složitosti programů zapsaných v běžných programovacích jazycích. V této kapitole jsem čerpal z [5, 17].

6.1 Teoretická vs. praktická řešitelnost problémů

V kapitole 5.2 jsme uvedli klasifikaci rozhodovacích problémů na ty, které jsou z teoretického pohledu (alespoň částečně) rozhodnutelné a ty, které jsou nerozhodnutelné. Nyní se blíže zaměříme na první uvedenou třídu problémů a představíme její další dělení v souvislosti s otázkou složitosti jejich rozhodování ([5]). Konkrétněji se budeme soustředit na srovnání problémů, které jsou ([17]):

- „Prakticky řešitelné“ – Jsou to problémy, u kterých požadavky na zdroje (doba výpočtu a/nebo potřebná paměť) programů, které je řeší, jsou takové, že je obecně lze splnit.
- „Prakticky neřešitelné“ – To jsou naopak problémy, u kterých požadavky na zdroje (doba výpočtu a/nebo potřebná paměť) programů, které je řeší, rostou s velikostí instance problému tak rychle, že je obecně nelze splnit pro větší instance problému.

V dalším textu budeme předpokládat, že pokud požadavky na zdroje programů rostou podle nějaké polynomiální funkce závislé na velikosti instance problému, pak považujeme tento problém za prakticky řešitelný. Naopak, pokud požadavky na zdroje porostou rychleji, pak považujeme tento problém za **prakticky** neřešitelný (s výjimkou velmi malých instancí problému).

6.2 Specifikace problémů pomocí jazyků

Aby bylo možné porovnávat velmi rozdílné druhy problémů z pohledu jejich složitosti, je třeba zavést jednotný způsob jejich popisu. Stejně jako v kapitole 5.2 budeme i zde předpokládat, že problémy mohou být specifikovány pomocí jazyků. Jako příklad si uveďme:

- $L_1 = \{w \in \{a, b\}^* \mid$ žádné dva po sobě jdoucí symboly řetězce w nejsou shodné$\}$ (regulární jazyk).

- $L_2 = \{a^i b^j c^k \mid i, j, k \geq 0 \wedge ((i \neq j) \vee (j \neq k))\}$ (bezkontextový jazyk).
- $L_3 = \{a^n b^n c^n \mid n \geq 0\}$ („jednoduchý“ jazyk, který není bezkontextový).
- $L_4 = SAT = \{\langle w \rangle \mid w$ je správně vytvořená formule Booleovské logiky a w je splnitelná$\}$ („složitý“ jazyk, který není bezkontextový).

6.3 Problémy při analýze složitosti programů

Obecně je cílem analýzy složitosti vyjádřit požadované zdroje (ať už čas nebo prostor) jako funkci závisející na délce vstupu.

Pokud chceme říci, že program, spuštěný na nějakém vstupu, provede p kroků nebo použije m paměťových míst, potřebujeme nejprve vědět, co se počítá jako krok a co jako paměťové místo. Uvažujme následující funkci `product()`, která vrací součin čísel na vstupu:

```
int product(int n,int A[n])
{
  int result=1;
  for(int i=0;i<n;i++)
  {
    result=result*A[i];
  }
  return result;
}
```

Předpokládejme, že `product()` je spuštěna na vstupním vektoru A o 10 číslech. Kolik kroků provede před tím, než bude ukončena? Při pokusu odpovědět na tuto otázku narazíme hned na několik problémů:

- Jak tuto hodnotu vůbec určit? Bude stačit zvýšit počet kroků o jedničku za každé provedení libovolného řádku kódu?
- Jak potom ale pracovat s řádkem obsahujícím `for`?
- Je správné uvažovat, že inkrementace indexové proměnné `i` je stejně náročná jako násobení dvou čísel?

Prozatím se dále tímto problémem nezabývejme a podívejme se raději na analýzu prostorové složitosti funkce `product()`. Zde se nabízí uvažovat následovně. Funkce `product()` potřebuje vlastně pouze dvě paměťová místa (navíc k paměti, která je použita k uložení vstupu) a to na uložení indexové proměnné `i` a výsledku násobení `result`. Z uvedeného plyne, že funkce `product()` požaduje pouze konstantní počet paměťových míst navíc (konkrétně 2), nezávisle na jejím vstupu. Nyní je ale potřeba si uvědomit, že velikost výsledku funkce roste s postupným násobením vstupním vektorem A. V tomto případě pak ale počet bitů potřebných pro uložení výsledku může růst spolu s rostoucím počtem položek ve vektoru A. Tímto se dostáváme do podobného problému jako při analýze časové složitosti, tj. co bychom tedy měli vlastně počítat?

Oba uvedené problémy se nám podaří vyřešit zvolením specifického modelu výpočtu. Bude jím Turingův stroj. Pak stačí pro určení časové složitosti spočítat počet kroků provedených Turingovým strojem a pro určení prostorové složitosti spočítat počet navštívených políček pásky. Analýzu složitosti programu pak budeme chápat jako analýzu složitosti výpočtů příslušného TS.

6.4 Analýza složitosti výpočtů Turingova stroje

Pro daný Turingův stroj M a daný vstupní řetězec w můžeme zjistit přesný počet kroků, které M provede, pokud je spuštěn s řetězcem w na svém vstupu. Stejně tak můžeme zjistit přesný počet políček pásky, které M navštíví během uvedeného výpočtu. Pro obecnější popis složitosti výpočtů daného TS M (na libovolném vstupu) definujeme dvě funkce: T_M pro určení časové a S_M prostorové složitosti.

Definice 6.1. Řekneme, že k-páskový DTS (resp. NTS) M přijímá jazyk L nad abecedou Σ v čase $T_M : \mathbb{N} \to \mathbb{N}$, jestliže $L = L(M)$ a M přijme (resp. může přijmout) každé $w \in L$ v nanejvýš $T_M(|w|)$ krocích.

Definice 6.2. Řekneme, že k-páskový DTS (resp. NTS) M přijímá jazyk L nad abecedou Σ v prostoru $S_M : \mathbb{N} \to \mathbb{N}$, jestliže $L = L(M)$ a M přijme (resp. může přijmout) každé $w \in L$ při použití nanejvýš $S_M(|w|)$ políček pásky.

Jak si lze všimnout, v obou definicích se jedná o analýzu nejhoršího případu, tedy např. funkce T_M vrací pro danou délku vstupu n počet kroků, který TS M provede v nejhorším případě (na vstupu o délce n, který je „nejtěžší“). Alternativně lze definovat funkce T_M a S_M tak, aby určovaly složitost nejlepšího nebo průměrného případu (obvykle se ale věnuje největší pozornost složitosti nejhoršího případu).

6.5 Automatizovaná analýza složitosti

Na začátku je třeba říci, že abychom byli schopni analyzovat složitost výpočtu Turingova stroje (resp. určitého algoritmu zapsaného v nějakém programovacím jazyce), je nejdříve nutné určit, zda daný TS někdy v budoucnu zastaví (resp. běh daného programu někdy skončí). Je tedy nejdříve nutné rozhodnout problém zastavení, což je ale (jak jsme si uvedli v kapitole 5) obecně nerozhodnutelný problém, a proto nelze provádět ani analýzu složitosti obecně libovolného TS (resp. programu). Prakticky se pro automatizovanou analýzu složitosti používají tzv. metriky, kdy se fundovaná uspořádání (viz kapitola 5.6) obohatí o metriky a během analýzy se pak sleduje nejen to, že se nějaká míra snížila, ale také jak moc. Jako příklad uveďme metriky pro programy s datovým typem `integer` ([18]).

Kapitola 7

Návrh aplikace

V této kapitole bude uveden návrh aplikace, která tvoří praktickou část práce. V návrhu se počítá s tím, že aplikace se bude skládat ze dvou částí – jádra aplikace a grafického uživatelského rozhraní (GUI), čemuž také odpovídá členění této kapitoly na podkapitoly. V každé z podkapitol pak bude popsán návrh odpovídající části aplikace.

7.1 Externí knihovna pro práci s grafy

V návrhu se při implementaci aplikace počítá s využitím existující knihovny, která bude v aplikaci použita pro vykreslování a práci s grafy. Tím máme na mysli, aby knihovna zejména podporovala/zajišťovala:

- Reprezentaci grafu ve vhodné datové struktuře.
- Čtení řetězců zapsaných v uzlech a na hranách grafu.
- Zjištění sousedních uzlů daného uzlu.
- Zjištění dvojice uzlů, které spojuje daná hrana.
- Změny stylů vykreslovaní uzlů a hran.

Jelikož tato část aplikace silně závisí na tom, které všechny uvedené požadavky bude knihovna splňovat, počítá se v návrhu jádra aplikace se dvěma variantami:

- Knihovna bude podporovat jen vykreslování grafů s velmi omezenou nebo žádnou možností další práce s grafy. V tomto případě bude nutné realizovat veškeré záležitosti ve vlastní režii.
- Knihovna bude podporovat jak vykreslování tak i další potřebné funkce pro práci s grafy. Zde bude postačovat vhodně využít poskytovaných funkcí knihovny.

V následujících podkapitolách bude představen návrh jádra aplikace pro obě uvedené varianty.

7.2 Návrh jádra aplikace

7.2.1 Realizace jádra aplikace zcela ve vlastní režii

Jádro aplikace je v tomto případě zamýšleno jako zcela nezávislé na grafickém uživatelském rozhraní a tedy musí obsahovat veškeré informace potřebné pro simulaci výpočtu Turingova stroje – reprezentaci Turingova stroje, jeho aktuální konfigurace a případně další informace. Uveďme si výčet toho nejdůležitějšího, co musí být uloženo v jádru aplikace:

- Samotný Turingův stroj – Pro reprezentaci TS nám postačí mít uloženu ekvivalentní reprezentaci kompozitního diagramu. Protože jde v zásadě o graf, stačí mít v jádru aplikace uloženy informace o uzlech a hranách vedoucích mezi těmito uzly.
  - Uzly – Budou reprezentovány pomocí pole řetězců, jehož indexy budou ID jednotlivých uzlů kompozitního diagramu. Na každém indexu pole pak bude uložen řetězec specifikující, které základní stavební bloky TS a v jakém pořadí mají být simulovány, pokud se simulace dostane do tohoto uzlu.
  - Hrany – Budou reprezentovány pomocí pole seznamů trojic. Pole bude obsahovat pro každý uzel kompozitního diagramu jeden seznam (pro daný uzel bude jeho seznam uložen na indexu pole rovném jeho ID – viz předchozí bod). Každý seznam pak obsahuje trojice, kde každá popisuje jednu odchozí hranu z odpovídajícího uzlu a sestává z:
    - Řetězce zapsaného u dané hrany – Specifikuje symboly, přes které je možné provést předání řízení.
    - Symbol proměnné parametrové konvence – Volitelně, jen pokud hrana využívá parametrové konvence.
    - ID cílového uzlu – ID uzlu, do kterého se při předání řízení pomocí této hrany dostaneme.
- Obsahy pásek – Budou reprezentovány pomocí pole řetězců, kde každý řetězec specifikuje aktuální obsah odpovídající pásky simulovaného Turingova stroje. Jak víme z předchozích kapitol, je každá páska TS směrem vpravo nekonečná, ale přitom platí, že na každém políčku, které nesloužilo k uložení některého ze symbolů vstupního řetězce TS a které nebylo TS dosud navštíveno, je zapsán symbol Δ. Z toho plyne, že obsah pásky můžeme reprezentovat jako řetězec, který zachycuje obsah pouze těch políček pásky, která na začátku simulace obsahovala symbol vstupního řetězce nebo na která TS už nějaký symbol zapsal. Jednodušeji řečeno, při zaznamenávání obsahu pásky budeme ignorovat nekonečně dlouhý řetězec symbolů Δ, který se nachází na pásce od určité pozice směrem vpravo.
- Pozice hlav – Budou reprezentovány jako pole přirozených čísel (i s nulou). Pro nejlevější pozici hlavy na pásce bude použito číslo 0.
- Stav TS – Bude reprezentován jako dvojice (ID uzlu, ID základního stavebního bloku TS v tomto uzlu). První složka byla vysvětlena výše, druhá složka je pak jednoduše označení základního stavebního bloku TS (jeho pořadí) v rámci uzlu. Tyto složky stavu TS budou v každém okamžiku specifikovat, simulace které části kompozitního diagramu byla právě dokončena. Zmíněná konvence bude použita proto, aby bylo možné vždy část kompozitního diagramu, u které byla aktuálně dokončena simulace,

vyznačit v GUI aplikace, aby byla simulace výpočtu TS více srozumitelná pro uživatele.

- Hodnoty proměnných parametrové konvence – Budou uloženy v poli indexovaném jmény těchto proměnných. Pro každou proměnnou bude na jejím indexu uložen symbol, který aktuálně tato proměnná obsahuje.

7.2.2 Realizace jádra aplikace s využitím knihovny

V tomto případě bude jádro aplikace silně navázáno na použitou knihovnu a tedy se dá očekávat, že i na grafické uživatelské rozhraní. Jelikož zde předpokládáme, že knihovna bude schopna zastoupit vše týkající se reprezentace Turingova stroje uvedené v předchozí kapitole, stačí abychom měli v jádru aplikace uložena data týkající se aktuální konfigurace TS a případné další informace. Obsahy pásek, pozice hlav stroje a hodnoty proměnných parametrové konvence budou reprezentovány tak, jak bylo popsáno v předchozí kapitole. Reprezentace stavu TS bude ale pozměněna ve smyslu, že zamýšlené „ID uzlu“ bude potřeba změnit na např. ukazatel na příslušný uzel kompozitního diagramu apod. (závisí na použité knihovně).

Jádro aplikace pak bude v obou případech poskytovat v zásadě jedinou funkci, která bude provádět simulaci jednoho kroku výpočtu zadaného Turingova stroje z aktuální konfigurace.

7.3 Grafické uživatelské rozhraní aplikace (GUI)

GUI aplikace bude buď fungovat jako nástavba nad jádrem aplikace a bude mu zasílat vstupy od uživatele a zároveň zobrazovat výsledky vypočtené jádrem aplikace uživateli (varianta z kapitoly 7.2.1) nebo bude s jádrem aplikace pevně provázáno (varianta z kapitoly 7.2.2). Aplikace se bude nacházet vždy v jednom ze dvou režimů – editačním nebo simulačním. V následujícím textu budou tyto režimy podrobněji vysvětleny i s ilustracemi, jak bude okno aplikace v daném režimu přibližně vypadat.

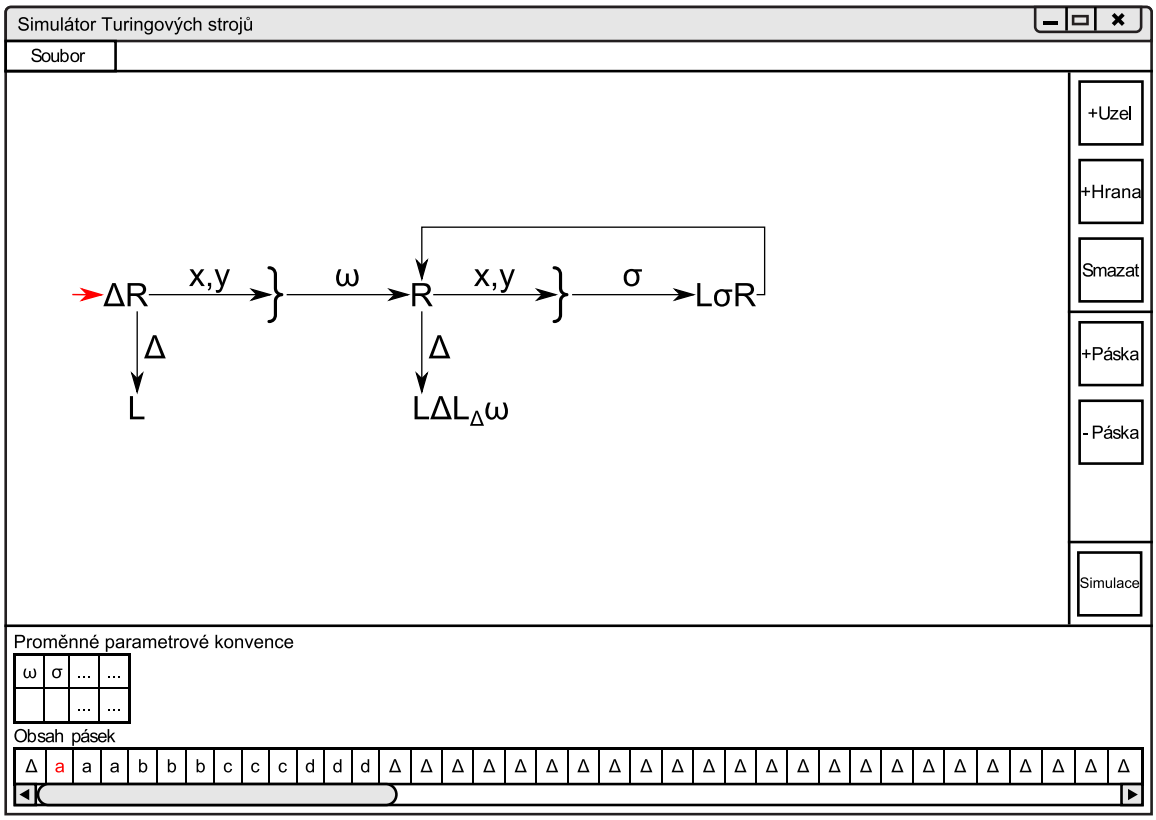
V každém z těchto režimů bude v aplikaci zobrazeno horní tlačítkové menu, ze kterého bude možné provádět operace jako založení nového, uložení nebo načtení kompozitního diagramu.

7.3.1 Editační režim

Jak už napovídá sám název, v tomto režimu bude možné do aplikace zadat kompozitní diagram TS, případně editovat již existující. Editační režim sám o sobě nebude umožňovat simulaci běhu zadaného TS, tu bude možné spustit až po přechodu do simulačního režimu. Návrh GUI aplikace v editačním režimu je na obrázku 7.1. GUI bude rozděleno do třech oddělených částí:

- Prostor s ovládacími tlačítky – Obsahuje tlačítka pro editaci kompozitního diagramu, zvolení požadovaného počtu pásek stroje a tlačítko pro přechod do simulačního režimu.

- Prostor pro zakreslení kompozitního diagramu TS – Bude zde vyznačen i počáteční uzel v kompozitním diagramu, ze kterého simulace začne.



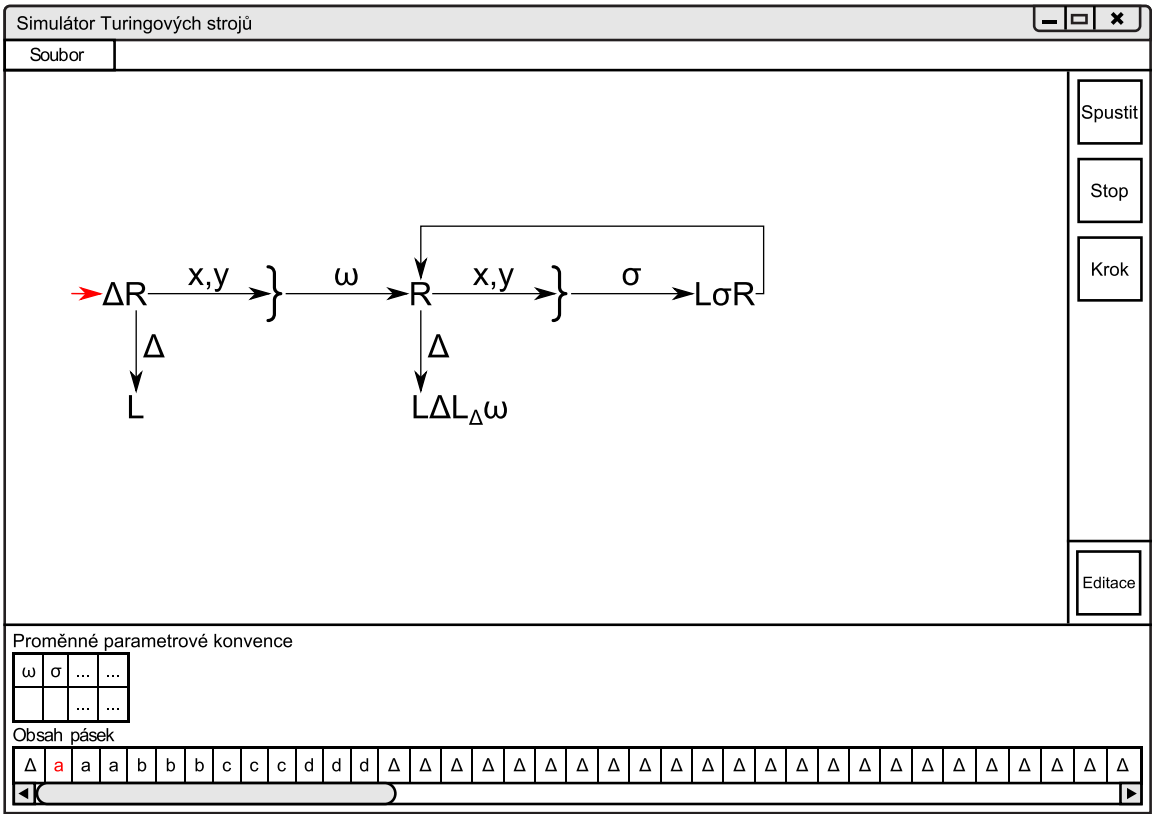
Obrázek 7.1: Návrh grafického uživatelského rozhraní aplikace – Editační režim

- Prostor s výpisem obsahů pásek a hodnot proměnných parametrové konvence – Tyto položky bude možné v editačním režimu zadávat a měnit včetně počátečních pozic hlav na páskách, které budou na každé pásce vyznačeny červeně zvýrazněným symbolem.

7.3.2 Simulační režim

V simulačním režimu bude značně omezena možnost měnit zadaný kompozitní diagram a další hodnoty, které bylo možné měnit v rámci editačního režimu. V tomto režimu bude aplikace umožňovat hlavně simulaci běhu Turingova stroje na zadané počáteční konfiguraci. Návrh GUI aplikace v simulačním režimu je na obrázku 7.2. GUI bude opět rozděleno do třech oddělených částí, z nichž dvě zůstávají téměř beze změny:

- Prostor s ovládacími tlačítky – Obsahuje tlačítka pro ovládání simulace běhu TS a tlačítko pro přechod do editačního režimu. Pro ovládání simulace zde konkrétně budou tlačítka pro:
  - Spuštění automatického krokování simulace.
  - Úplné zastavení simulace a navrácení se ke konfiguraci TS, která byla zadána v editačním režimu.
  - Manuální krokování simulace.



Obrázek 7.2: Návrh grafického uživatelského rozhraní aplikace – Simulační režim

Po spuštění automatického krokování simulace pomocí tlačítka „Spustit“ se na jeho místě objeví tlačítko „Pozastavit“ pro pozastavení simulace. Pokud je simulace pozastavena, objeví se opět standardní sada tlačítek z předcházejícího výčtu s tím, že tlačítko „Spustit“ má nyní význam pokračování v automatické simulaci.

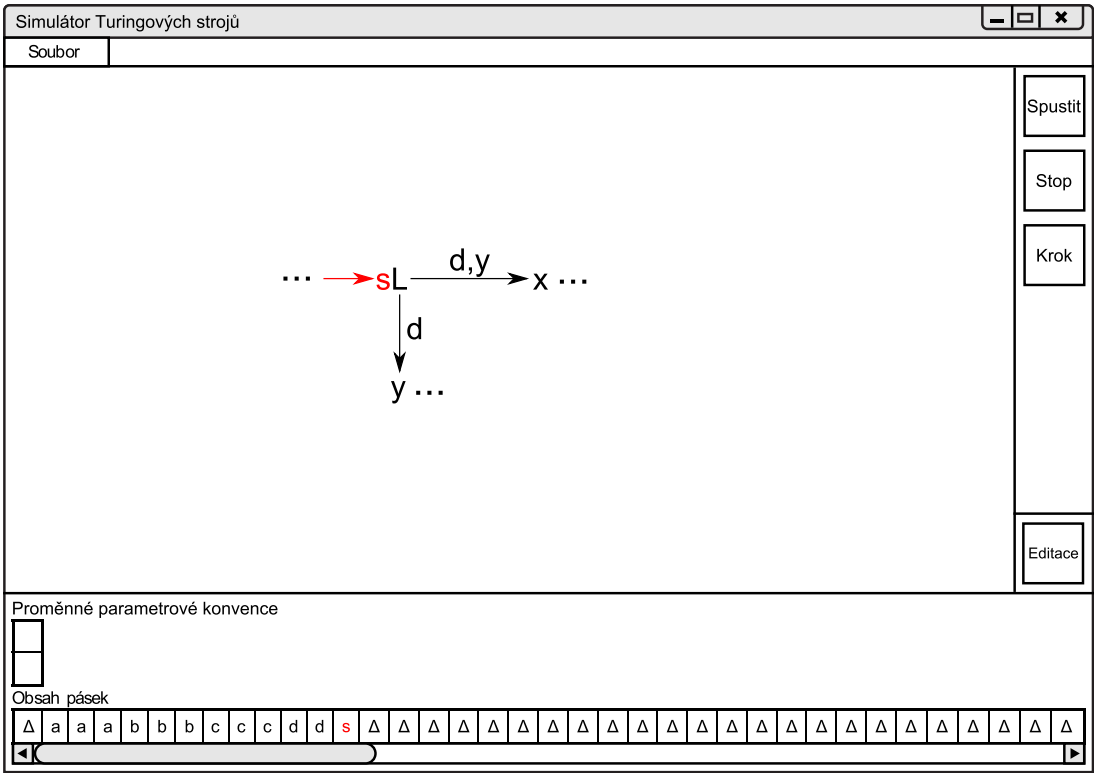
- Prostor s kompozitním diagramem TS – Obsahuje zadaný kompozitní diagram s barevně vyznačeným místem, jehož simulace byla dokončena v předchozím kroku.
- Prostor s výpisem obsahů pásek, aktuálních pozic hlav na páskách (vyznačeny pomocí červeně zvýrazněného symbolu) a hodnot proměnných parametrové konvence. Tyto položky bude možné zadávat a měnit i v simulačním režimu.

V simulačním režimu dále nastává problém, pokud je zadaný TS nedeterministický. Samotná aplikace totiž nemá šanci vybrat správný krok, kterým by se měl výpočet vydat, aby se v případě, že existuje nějaká posloupnost kroků vedoucí k přijetí vstupního řetězce, také tato posloupnost nalezla. Volba kroku, který se provede jako následující, bude tedy v těchto případech přenechána na uživatele.

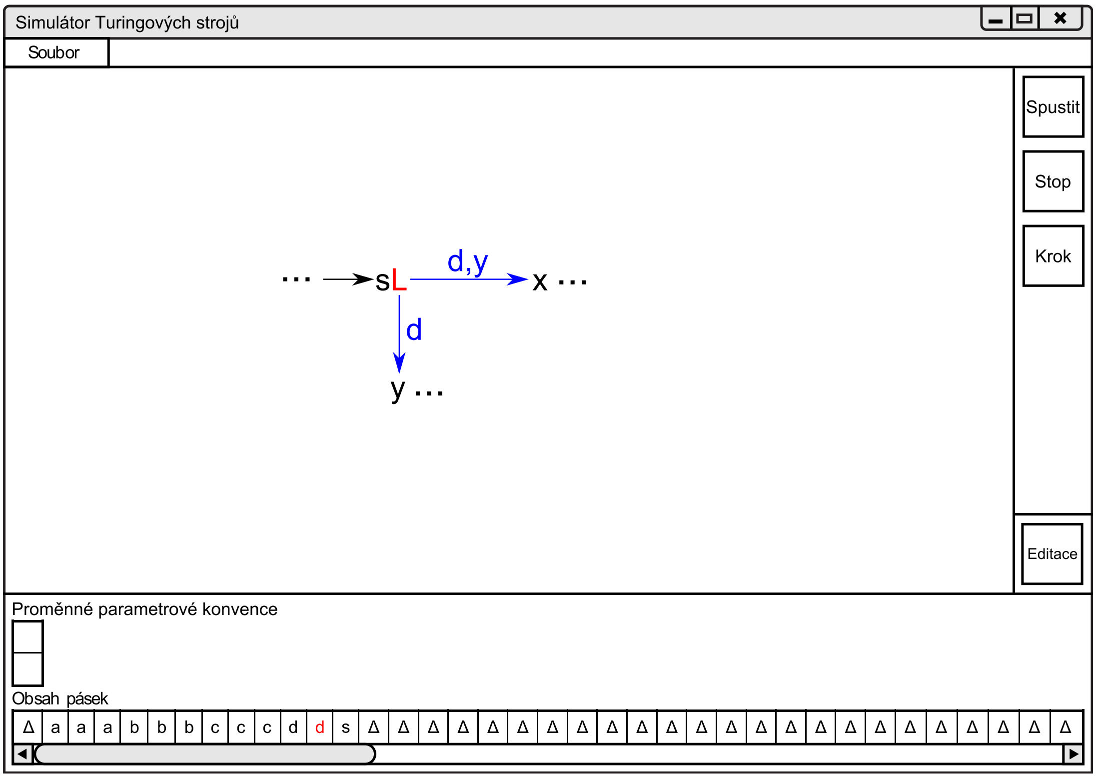
7.3.2.1 Příklad simulace výpočtu Turingova stroje

Na závěr kapitoly o návrhu si uveďme krátký příklad, jak by v aplikaci mohla fungovat simulace výpočtu TS. Předpokládejme, že jsme se v simulaci dostali do stavu uvedeného

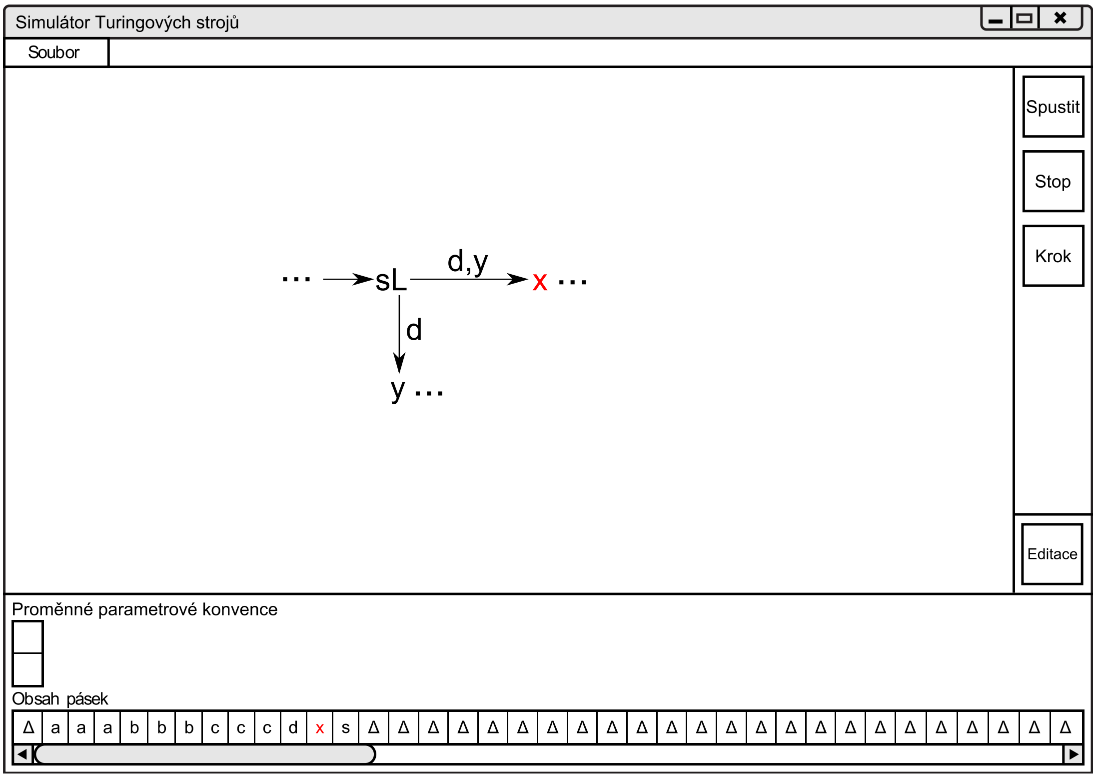
na obrázku 7.3 (v předchozím kroku simulace byl zapsán symbol s na pásku). Po provedení následujícího kroku bude na základě stroje L proveden posuv hlavy na pásce vlevo a v kompozitním diagramu bude červeně vyznačen stroj L. Dále, protože se na aktuálním políčku pásky vyskytuje symbol d a pro tento symbol existuje v kompozitním diagramu více možností přechodu, dá aplikace uživateli na výběr (ve formě modrého zbarvení hran), který z těchto přechodů má být proveden (obrázek 7.4). Předpokládejme, že uživatel zvolil přechod směrem vpravo. Výsledkem bude, že se na políčko pásky umístěné pod hlavou TS zapíše symbol x a patřičně se aktualizuje i obarvení v kompozitním diagramu (obrázek 7.5).



Obrázek 7.3: Výchozí konfigurace TS



Obrázek 7.4: Konfigurace TS po provedení jednoho kroku výpočtu



Obrázek 7.5: Konfigurace TS po provedení celkem dvou kroků výpočtu

Kapitola 8

Implementace

V této kapitole bude stručně shrnut popis implementace výsledné aplikace. Cílem je poskytnout čtenáři základní přehled o principech fungování aplikace a nastínit řešení některých jejích částí. Jako implementační jazyk byla zvolena Java s využitím GUI toolkitu AWT/Swing. Dále byly také použity následující knihovny:

- JGraphX[1] – Nejdůležitější knihovna, poskytuje rozhraní pro práci s grafy a umožňuje jejich vykreslování a editaci v okně aplikace.
- Apache Commons IO[2] – Použita pro načítání souborů do paměti v požadovaném formátu.
- Apache Commons Lang[3] – Použita pro konverzi mezi HTML znakovými entitami a symboly, které reprezentují.

8.1 Celkový popis implementace aplikace

Celá aplikace je implementována v sedmi třídách. Metoda `main()`, která je zavolána po spuštění aplikace, je umístěna ve třídě `TuringSimulator`. Implementace aplikace je logicky rozdělena na tři části:

- Implementace vlastního simulátoru Turingova stroje včetně detekce cyklení (třída `TuringCore`).
- Implementace GUI aplikace (zde řadíme třídy `GUIObj`, `TmxCellEditor`, `TmxGraph`, `TmxGraphComponent`).
- Implementace heuristiky, jejíž účelem je snaha detekovat na základě analýzy kompozitního diagramu Turingova stroje, zda tento stroj vždy zastaví, nebo nejsme schopni odpovědět (dále také jako „Detekce zastavení“) – pro více informací o problému zastavení TS viz kapitola 5 (do této části patří jen třída `HaltingDetectionObj`).

Přestože se jedná o části, kde každá zajišťuje odlišné funkce aplikace, jsou navzájem silně propojené a nelze je bez větších zásahů do jejich zdrojových kódů provozovat samostatně. Potřeba propojení GUI a simulátoru je uvedena již v kapitole o návrhu (7.2.2) a plyne z toho,

[1] Dostupné na: https://github.com/jgraph/jgraphx.
[2] Dostupné na: http://commons.apache.org/proper/commons-io/.
[3] Dostupné na: http://commons.apache.org/proper/commons-lang/.

že samotná knihovna JGraphX (která je využívána zejména grafickým uživatelským rozhraním aplikace) je sama schopna kompletně reprezentovat kompozitní diagram Turingova stroje a tudíž je zbytečné, aby jeho kopie byla ještě uložena a udržována v části implementující simulátor. Položky reprezentující ať už samotný TS, či jeho konfiguraci (obsahy pásek, pozice hlav, stav TS – tj. aktuálně odsimulovaný základní stavební blok TS v kompozitním diagramu, hodnoty proměnných parametrové konvence) jsou proto rozděleny mezi třídy `GUIObj` a `TuringCore` tak, aby se nacházely vždy v místě, kde budou nejčastěji používány. Ostatní části aplikace pak s nimi pracují zprostředkovaně a tudíž není potřeba nikde udržovat shodnou druhou kopii stejné informace. Podobná situace je i v případě propojení třídy `HaltingDetectionObj` se zbytkem aplikace. Zde je nutnost propojení ještě více zřejmá – detekce zastavení je prováděna na Turingově stroji, který není spuštěn a zároveň analýza neprovádí v reprezentaci TS žádné změny. Bylo by proto zcela zbytečné provádět kopírování potřebných informací do třídy provádějící analýzu a zde je využívat. Analýza TS prováděná třídou `HaltingDetectionObj` probíhá tedy přímo s využitím dat uložených v ostatních třídách aplikace.

Jak už bylo zmíněno, běh aplikace začíná v metodě `main()`, která je umístěna ve třídě `TuringSimulator`. Tato třída slouží hlavně pro vytvoření instancí tříd `TuringCore` a `GUIObj` a zavolání metody zajišťující vykreslení grafického uživatelského rozhraní aplikace (metoda `createGUI()`). Po vykreslení GUI čeká aplikace na to, až uživatel provede nějakou akci v okně aplikace. Akce jsou obsluhovány metodami ve třídě `GUIObj`, které ale mohou využívat i metody z ostatních tříd. Nejdůležitější metodou třídy `TuringSimulator`, kterou třída `GUIObj` využívá, je nepochybně metoda `step()`, která provádí simulaci jednoho kroku výpočtu Turingova stroje. Metoda `step()` při výpočtu pak zase zpětně využívá aktuální informace týkající se samotného Turingova stroje a jeho konfigurace poskytované třídou `GUIObj`. Aplikace tudíž umožňuje uživateli měnit některé vlastnosti týkající se chování a konfigurace Turingova stroje i během samotné simulace. Výsledek jednoho kroku výpočtu TS je navíc možné buď okamžitě zobrazit v okně aplikace (ve formě aktualizované konfigurace TS), nebo je možné ho zatím nezobrazovat a provést simulaci dalšího kroku výpočtu TS. Výsledek je tedy možné zobrazit až po simulaci několika kroků, což se využívá v případě, kdy uživatel v aplikaci zvolil, že má být prováděn větší počet kroků výpočtu najednou. V tomto případě tedy nedochází ke zbytečnému obnovování zobrazené aktuální konfigurace TS[4] v okně aplikace pro každý simulovaný krok výpočtu TS, ale až pro ten poslední z dané sekvence. V případě, že je v aplikaci spuštěna detekce zastavení, je vždy vytvořena nová instance třídy `HaltingDetectionObj` a spuštěna její hlavní metoda `run()`. Výsledek provedené analýzy je pak uživateli zobrazen v okně aplikace. Zbývající třídy `TmxCellEditor`, `TmxGraph` a `TmxGraphComponent` jsou relativně velmi jednoduché a z celkového pohledu na fungování aplikace ne tak důležité jako právě představené třídy, jejich bližší popis je proto uveden až dále.

V následujících kapitolách budou postupně představeny všechny třídy a popsány jejich nejdůležitější metody. Předtím ještě dodejme, že celá aplikace běží v jednom vlákně, které se v Javě nazývá Event Dispatch Thread (EDT). EDT je původně určeno pouze pro spouštění částí programu určených pro manipulaci s GUI. Ostatní části programu by tedy měly být spouštěny v separátním vlákně, aby nedošlo k tomu, že EDT bude zahlceno prováděním těchto s GUI nesouvisejících částí a tudíž GUI aplikace přestane reagovat na uživatelovy podněty. Protože se ale v implementaci vytvořené aplikace nevyskytují žádné natolik náročné části, aby je EDT nestihlo zpracovat, rozhodl jsem se vše implementovat v tomto

[4]Konfigurace TS, se kterou pracuje třída `TuringSimulator` je samozřejmě vždy aktuální bez ohledu na konfiguraci zobrazovanou uživateli.

vlákně. Dalším důvodem bylo také to, že vícevláknové aplikace jsou obecně výrazně náročnější na odladění všech chyb a často je potřeba také řešit synchronizaci spolupracujících vláken.

8.2 Detailnější popis implementace jednotlivých tříd

V této podkapitole bude uvedena ke každé z tříd aplikace vždy její krátká charakteristika se zaměřením na to, co je hlavním účelem dané třídy a dále popis jejích důležitých metod a použitých principů.

8.2.1 Třída `TuringSimulator`

Jde o velmi jednoduchou třídu, která obsahuje pouze metodu `main()` a její účel byl z velké části již popsán v kapitole 8.1. Jedním z dalších úkolů této třídy je ještě také zavedení klávesových zkratek, které je možné následně v aplikaci využívat (více viz kapitola A.2).

8.2.2 Třída `GUIObj`

V této třídě je až na některé drobnosti implementováno celé GUI aplikace, zbylé části jsou ve třídách `TmxCellEditor`, `TmxGraph` a `TmxGraphComponent`. Pro vytvoření GUI v Javě je potřeba v zásadě naimplementovat dvě věci:

1. Metody pro vytvoření vlastního okna aplikace, komponent umístěných v něm (tlačítka, tabulky, zatrhávací políčka, menu, apod.), rozmístění těchto komponent a nastavení, jak se mají chovat při změně velikosti okna aplikace.

2. Metody obsluhující akce prováděné uživateli na těchto komponentách – co se má stát, když uživatel:

   - klikne na tlačítko, tabulku, příp. jinam,
   - stiskne nějakou klávesu na klávesnici,
   - vybere položku v menu,
   - edituje tabulku,
   - zavře okno aplikace,
   - apod.

   Konkrétněji jsou v Javě tyto akce obsluhovány tzv. posluchači (angl. listener). Jde o speciální třídy implementující definované rozhraní, které jsou přidruženy k určité komponentě GUI jako její posluchači (každá komponenta může mít více posluchačů). Jejich účelem je „poslouchat“, zda uživatel na dané komponentě neprovedl nějakou akci a pokud ano, spustí příslušnou metodu implementovanou pro obsluhu této akce. Např. posluchač přiřazený k určitému tlačítku, sledující kliknutí na toto tlačítko (nejčastější, ale lze sledovat i jiné akce), má jedinou metodu `actionPerformed()`, která je zavolána právě při kliknutí na tlačítko.

Nyní už přejdeme k popisu vybraných metod a posluchačů implementovaných v této třídě.

8.2.2.1 Metoda `savePrompt()`

Je důležitá v situacích, kdy uživatel vytvořil nebo editoval kompozitní diagram a bez uložení provedl akci, která pokud by se provedla, tak by znamenala smazání vytvořeného diagramu (zavření aplikace, otevření jiného nebo vytvoření nového kompozitního diagramu). Metoda `savePrompt()` slouží k tomu, aby ještě před provedením dané akce zobrazila dotaz uživateli, zda chce rozpracovaný kompozitní diagram uložit nebo ne. Pro realizaci vlastního uložení používá metodu `saveToFile()`. Pokud v diagramu k žádným změnám (od předchozího načtení resp. uložení) nedošlo, není dotaz na uložení uživateli zobrazen (využívá se příznaku `tChanged`, viz kapitola 8.2.2.6).

8.2.2.2 Posluchač `ShowHelpListener`

Obstarává vytvoření okna s nápovědou tak, aby jej bylo možné mít otevřené spolu s oknem aplikace a přecházet mezi nimi. Také provádí nastavení všech dalších požadovaných vlastností okna nápovědy.

8.2.2.3 Posluchač `HaltingDetectionListener`

Obsluhuje kliknutí na menu pro spuštění detekce zastavení TS. Jak už bylo zmíněno v kapitole 8.1, je za tímto účelem vytvořena nová instance třídy `HaltingDetectionObj` a spuštěna její hlavní metoda `run()`.

8.2.2.4 Metoda `createGUI()`

Jedna z nejdůležitějších metod, provádí vše uvedené v bodě 1 výše. Uveďme zkrácený výčet prováděných akcí:

- Vytvoření okna aplikace a nastavení jeho velikosti.
- Vytvoření horního tlačítkového menu.
- Nastavení atributů objektů knihovny JGraphX, aby se graf (kompozitní diagram) při interakci choval korektně a také aby se správně vykresloval.
- Vytvoření všech komponent zobrazených v okně aplikace s požadovanými vlastnostmi, nastavení jejich pozic a chování při změně velikosti okna aplikace pomocí tzv. layout managerů.

Dále se v této metodě ještě provádí inicializace všech používaných proměnných pomocí metody `newTS()` a následně se aplikace přepne do editačního režimu s využitím metody `SwitchToEditMod()`. Nakonec je celé takto vytvořené okno aplikace zobrazeno uživateli.

8.2.2.5 Metoda `createTapeContentTable()`

Vytváří tabulku zobrazující obsahy pásek TS. U této tabulky je nejprve potřeba vytvořit a nastavit vlastní třídu obstarávající vykreslování jednotlivých buněk tabulky (v Javě jde o třídu zvanou `DefaultTableCellRenderer`), protože se v tabulce musí nějakým způsobem vyznačit aktuální pozice hlavy stroje. Dále je ještě potřeba nastavit příslušné posluchače pro obsluhu kliknutí na tabulku a některé další vlastnosti tabulky.

8.2.2.6 Metoda `setTChanged()`

Jde o velmi jednoduchou metodu, která v zásadě jen nastavuje příznak `tChanged`, že byla v aktuálním kompozitním diagramu provedena nějaká změna uživatelem. Tento příznak je pak využíván v metodě `savePrompt()` – viz kapitola 8.2.2.1.

8.2.2.7 Metoda `newTS()`

Jak už napovídá název, jde o metodu, která provádí reinicializaci všech proměnných používaných pro uložení vlastního TS a jeho konfigurace. Je používána ve všech případech, kdy je potřeba zahodit aktuální TS a připravit aplikaci na nový.

8.2.2.8 Metoda `isTSValid()`

Metoda `isTSValid()` je volána před tím, než aplikace přejde do simulačního režimu a jejím účelem je zkontrolovat, zda zadaný TS splňuje jistá omezení. Pokud ne, je uživateli zobrazen dialog s popisem, kde se v zadaném TS nachází chyba, aby ji mohl snadno opravit, a přechod do simulace se neuskuteční. Mezi možné chyby patří např.:

- V uzlu kompozitního diagramu není uveden žádný řetězec.
- Speciální uzel parametrové konvence má pouze hranu, která do něj vstupuje.
- V kompozitním diagramu se vyskytuje číslo pásky, která dosud neexistuje.
- Za speciálním strojem „D“ (stroj, který okamžitě odmítne svůj vstup) se nachází nějaké další stroje – jde o chybu, protože by se na ně simulace nikdy nedostala.
- Z uzlu, který obsahuje stroj „D“, vedou nějaké odchozí hrany (předání řízení) – stejný problém jako v předchozím bodě.
- Neplatný formát řetězce uvedeného na některé hraně.

8.2.2.9 Metoda `SwitchToSimMod()`

Slouží pro přepnutí aplikace do simulačního režimu. Na začátku kontroluje pomocí metody `isTSValid()` platnost zadaného TS. Pokud je vše v pořádku, provede záměnu zobrazených tlačítek v pravé části okna aplikace, zakáže manipulaci s kompozitním diagramem TS a provede zálohu všech proměnných, jejichž obsah se může v simulaci změnit, ale u kterých se chceme při zpětném přechodu do editačního režimu vrátit k jejich původním hodnotám (např. obsahy pásek, pozice hlav na páskách, apod.).

8.2.2.10 Metoda `SwitchToEditMod()`

Provádí opačné akce, než metoda `SwitchToSimMod()`. Konkrétně navrátí do pravé části okna aplikace původní tlačítka, povolí provádět změny v kompozitním diagramu TS a obnoví hodnoty všech zazálohovaných proměnných (s využitím metody `restoreAll()`).

8.2.2.11 Metoda `restoreAll()`

Jak už bylo zmíněno, metoda provádí obnovu hodnot vybraných proměnných ze záloh, které vytváří metoda `SwitchToSimMod()`.

8.2.2.12 Metoda `ActualizeStateInGraph()`

Metoda slouží pro zobrazení aktualizovaného stavu TS (po provedení jednoho kroku výpočtu pomocí metody `nextStep()`) v okně aplikace. Stav TS je zobrazován ve formě červeně obarvených částí uzlů a hran kompozitního diagramu TS, tak jak bylo uvedeno v návrhu aplikace (viz kapitola 7.3.2). Vyznačení stavu se řídí proměnnými `stateCell` a `stateBB`, kde první určuje uzel, do kterého dospěla simulace v právě dokončeném kroku a druhá určuje pozici základního stavebního bloku TS, jehož simulace byla právě dokončena (v uzlu `stateCell`). Dodejme, že se v této metodě také provádí obarvení hran na modro v případě, že simulace dojde do bodu, kde je možné předat řízení více různým strojům (viz kapitola 7.3.2).

8.2.2.13 Metody `cStringToHTML()` a `HTMLTocString()`

Při návrhu a implementaci byla snaha, aby zobrazení kompozitních diagramů v aplikaci bylo co nejvěrnější způsobu, jakým se standardně zapisují. Zejména zde narážíme na to, že se v řetězcích zapisovaných v uzlech a na hranách standardně využívá horních a dolních indexů. Pro realizaci těchto stylů písma je v aplikaci využita možnost knihovny JGraphX zapsat text do uzlu i hrany v jazyku HTML. Protože by ale psaní HTML značek bylo pro uživatele (alespoň během editace kompozitního diagramu) velmi nepřehledné, je v aplikaci zaveden jiný způsob zadávání těchto stylů. Konkrétně pro zadání horního indexu se využívá znaku stříšky (^), který může být v případě víceznakového čísla doplněn o složené závorky (např. R^2 nebo R^{10} apod.). Pro zadání dolního indexu se používá znak podtržítka (_). Pro automatickou konverzi mezi těmito formáty jsou právě určeny metody `cStringToHTML()` a `HTMLTocString()`. K odpovídajícím konverzím dochází automaticky při zahájení (resp. ukončení) editace uzlu nebo hrany transparentně pro uživatele.

Metoda `cStringToHTML()` je značně složitější než její protějšek, protože dostává na vstup obecně řetězec zadaný uživatelem, který nemusí být v očekávaném formátu. Pro zpracování jsou využívány metody třídy `TuringCore`, konkrétně je zde použita metoda `parseNextMachine()` pro zpracování řetězců v uzlech a metoda `parseNextString()` pro hrany. Uvedené metody byly původně určeny pouze pro zpracovávání již ověřených platných řetězců pro účely simulace kroku výpočtu TS, ale nakonec se po malých úpravách osvědčily i pro tento úkol. Navíc tak došlo ke sjednocení metod, které provádějí zpracování řetězců pro účely simulace a zobrazení těchto řetězců uživateli (v dřívějších implementacích docházelo k nekonzistencím).

Metoda `HTMLTocString()` je naopak velmi jednoduchá, protože dostává na vstup řetězec, který už byl zpracován metodou `cStringToHTML()`. Postačí si tak s nahrazováním podřetězců, které vyhoví daným regulárním výrazům.

8.2.2.14 Metoda `nextStep()`

Metoda `nextStep()` provádí volání metody `step()` (tj. simulace dalšího kroku výpočtu TS) třídy `TuringCore` tolikrát, kolik uživatel nastavil v okně aplikace. Provádění metody `step()` může skončit dvěma způsoby: standardním návratem a nebo vyvoláním výjimky. Pokud metoda `step()` vyvolá výjimku, pak je tato v metodě `nextStep()` ošetřena ve formě vypsání dialogu uživateli a simulace je ukončena (a resetována). Případy, kdy metoda `step()` vyvolá výjimku, jsou uvedeny v kapitole 8.2.3.13. Standardní návrat značí úspěšné provedení simulace dalšího kroku výpočtu TS. Z návratové hodnoty je dále možné zjistit (jen v případě, že je aktivní funkce detekce cyklení TS), zda bylo detekováno cyklení TS v cyklu,

ze kterého je možné se vymanit. Jde o případ, kdy v některém uzlu na detekovaném cyklu bylo nedeterministicky rozhodnuto jít určitou cestou – pokud bychom se v tomto uzlu ale příště rozhodli jinak, je možné, že by se TS z cyklu dostal. V tomto případě je umožněno v simulaci pokračovat, jen je uživateli zobrazen příslušný varovný dialog. Vždy platí, že pokud nedojde k výjimce a (případně) opakovaná volání metody `step()` byla dokončena, je na závěr provedeno zobrazení aktualizovaného stavu TS v GUI aplikace s využitím metody `ActualizeStateInGraph()`.

8.2.2.15 Metoda `resetSim()`

Tato metoda provádí nejprve pozastavení automatické simulace (pokud běží) a následně pomocí metody `restoreAll()` obnoví TS do podoby, jak byl zadán v editačním režimu včetně jeho konfigurace.

8.2.2.16 Metoda `autoSim()`

Opět jde o velmi jednoduchou metodu, která pouze vytvoří nový časovač s dobou opakování danou intervalem nastaveným v aplikaci uživatelem. Po vypršení časovač volá metodu `nextStep()`.

8.2.2.17 Metody `ParConvTableAddNewVariable()`, `ParConvTableDeleteVariable()`, `ParConvTableSetVariableTo()`, `ParConvTableGetVariableVal()`, `parConvUnsetVariables1()` a `parConvUnsetVariables2()`

Jde o sadu metod usnadňující práci s proměnnými parametrové konvence. Hodnoty proměnných jsou ukládány přímo v tabulce k tomu určené a dále uvedené metody tedy pracují přímo s touto tabulkou. Funkce metod je většinou zřejmá již z jejich názvu:

- `ParConvTableAddNewVariable()` – Přidá proměnnou parametrové konvence s daným jménem.
- `ParConvTableDeleteVariable()` – Odstraní danou proměnnou parametrové konvence.
- `ParConvTableSetVariableTo()` – Uloží do dané proměnné daný symbol.
- `ParConvTableGetVariableVal()` – Vrátí aktuální symbol uložený v dané proměnné.
- `parConvUnsetVariables1()` a `parConvUnsetVariables2()` – Jde vlastně o jedinou metodu, která je z jistých důvodů rozdělena na dvě. Společně provádí odstranění jednoho výskytu proměnné parametrové konvence (jedna proměnná se může vyskytovat v kompozitním diagramu vícekrát a pak je nutné ji z tabulky proměnných odstranit až v případě, kdy se v diagramu vůbec nevyskytuje). Pokud byl daný výskyt posledním, provedou úplné odstranění zpracovávané proměnné pomocí metody `ParConvTableDeleteVariable()`.

8.2.2.18 Metoda `ParConvSetVariables()`

Tato metoda slouží pro zobrazení dialogu, který umožňuje uživateli zadat řetězec obsahující jména proměnných parametrové konvence, který má být umístěn na hraně, která vede ze speciálního uzlu parametrové konvence (metoda je volána v případě editace nebo vytvoření takové hrany). Uvedený řetězec standardně slouží pro určení, do kterých proměnných mají být uloženy aktuálně čtené symboly ze kterých pásek. Správnost zadaného řetězce je okamžitě kontrolována a není možné zadat neplatné hodnoty (narozdíl třeba od ostatních řetězců zapisovaných do kompozitního diagramu). Potřeba zadaný řetězec tímto způsobem kontrolovat plyne z toho, že na základě těchto řetězců je okamžitě (ještě během editace) vytvářena tabulka proměnných parametrové konvence. Tabulka se vytváří již během editace kompozitního diagramu zejména proto, aby bylo možné proměnným parametrové konvence nastavit jejich hodnoty případně ještě před spuštěním simulace. Není tak přípustné, aby byly řetězce specifikující jména proměnných neplatné. Pokud uživatel zadá do zobrazeného dialogu neplatný řetězec, nedojde k uzavření dialogu a uživatel musí řetězec opravit. Jakmile je vložen platný řetězec, jsou jména zadaných proměnných přidána do tabulky proměnných parametrové konvence pomocí metody `ParConvTableAddNewVariable()`, dialog je uzavřen a zadaný řetězec je zapsán k dané hraně kompozitního diagramu.

8.2.2.19 Posluchač `ConnectListener`

Tento posluchač je spouštěn při každém připojení hrany k uzlu, jeho hlavním účelem je kontrolovat, zda je nově vytvořené připojení platné. Jde zejména o zajištění, aby propojení speciálních uzlů parametrové konvence odpovídalo způsobu, jakým se standardně zakresluje v kompozitních diagramech (tj. maximálně jedna příchozí a jedna odchozí hrana). Potřeba okamžité kontroly správnosti zde má stejné opodstatnění jako v kapitole 8.2.2.18.

8.2.2.20 Posluchač `MainMouseListener`

Posluchač `MainMouseListener` slouží pro ošetřování kliknutí myši do prostoru určeného pro zakreslení kompozitního diagramu TS. Posluchač sleduje, zda při kliknutí nebyla současně stisknuta některá z kláves Alt nebo Shift a pokud ano, zajistí, aby kliknutí provedlo v kompozitním diagramu příslušné akce (viz kapitola A.2). Dále také vytváří kontextová menu zobrazovaná při kliknutí pravým tlačítkem myši na uzel nebo hranu kompozitního diagramu.

8.2.2.21 Metoda `CRC32()`

Tato metoda slouží pro vytvoření a případně ověření správnosti kontrolního součtu, který je součástí souborů, do kterých jsou ukládány TS vytvořené v aplikaci.

8.2.2.22 Metoda `openFile()`

Metoda provádí načítání dříve vytvořeného TS ze souboru. Na začátku nejprve vypočítá kontrolní součet souboru a zkontroluje zda odpovídá kontrolnímu součtu uloženému v souboru (vše s využitím metody `CRC32()`). Pokud ne, je načtení TS ze souboru přerušeno. Jinak metoda provede nastavení příslušných proměnných aplikace dle hodnot uvedených ve zpracovávaném souboru a přepne aplikaci do editačního režimu pomocí metody `SwitchToEditMod()`.

8.2.2.23 Posluchač `OpenItemListener`

Tento posluchač řeší situaci, kdy uživatel kliknul na tlačítko „Otevřít“ v menu aplikace. Na začátku vytvoří a následně zobrazí okno pro výběr souboru, který má být otevřen. Po vybrání souboru zavolá metodu `savePrompt()` pro případné uložení rozpracovaného TS a následně volá metodu `openFile()` pro realizaci vlastního otevření souboru. Pokud se otevření nezdařilo, zobrazí uživateli dialog s dalšími informacemi.

8.2.2.24 Metoda `saveToFile()`

Metoda provádí ukládání TS, který je v aplikaci aktuálně vytvořen, do daného souboru. Nejprve provede uložení hodnot vybraných proměnných do souboru v předepsaném formátu, následně vypočítá pomocí metody `CRC32()` kontrolní součet a připojí ho na konec souboru.

8.2.2.25 Metoda `showSaveAsDialog()`

Metoda vytvoří a následně zobrazí okno pro zadání názvu souboru, do kterého má být aktuálně vytvořený TS uložen. Jakmile uživatel úspěšně zadá název souboru, je zavolána metoda `saveToFile()` pro realizaci vlastního uložení TS.

8.2.3 Třída `TuringCore`

Ve třídě `TuringCore` jsou naimplementovány veškeré metody týkající se simulace výpočtu Turingova stroje. Jak už ale bylo popsáno v kapitole 8.1, nejsou zde uloženy veškeré informace o Turingově stroji a jeho konfiguraci, k některým datům je tak potřeba přistupovat přes třídu `GUIObj`. Nyní už přejdeme k popisu důležitých metod implementovaných v této třídě.

8.2.3.1 Metoda `moveHead()`

Jde o pomocnou metodu, která na dané pásce provádí posuv hlavy Turingova stroje o jedno políčko pásky směrem vlevo nebo vpravo. Metoda také sleduje, zda hlava stroje „nepřepadla“ vlevo, a pokud ano, vyvolá výjimku (mechanismus vyvolávání výjimek ve třídě `TuringCore` je detailněji popsán v kapitole 8.2.3.13).

8.2.3.2 Metoda `isFeasible()`

Metoda `isFeasible()` určuje, zda je možné v simulaci z aktuální konfigurace TS provést přechod pomocí hrany, u které je uveden daný řetězec. Pro rozhodnutí, zda přechod bude možné provést nebo ne, využívá tato metoda standardní sémantiku řetězců zapisovaných u hran, používanou v kompozitních diagramech:

- Pokud není u hrany uveden žádný řetězec, je automaticky přechod možné provést.
- Pokud je u hrany zapsán řetězec, který nezačíná symbolem negace, pak musí být na jednotlivých páskách TS aktuálně čtena alespoň jedna z kombinací symbolů uvedených v tomto řetězci (kombinace jsou odděleny čárkami). Pokud je součástí kombinace symbolů proměnná parametrové konvence, pak se (namísto symbolu reprezentujícího jméno této proměnné) musí na aktuálně čteném políčku odpovídající pásky TS vyskytovat symbol, který je uložen v této proměnné.

- V případě, kdy řetězec u hrany začíná symbolem negace, je význam přesně opačný. Zde, aby bylo možné daný přechod provést, nesmí být na páskách TS aktuálně čtena žádná z kombinací symbolů uvedených v tomto řetězci a pokud je součástí kombinace symbolů proměnná parametrové konvence, pak se nesmí na aktuálně čteném políčku odpovídající pásky TS vyskytovat symbol, který je uložen v této proměnné.

Ke zpracování řetězců je zde využita metoda `parseNextString()` (viz kapitola 8.2.3.4).

8.2.3.3 Metoda `parseTapeNum()`

Tato metoda je používána v metodách `parseNextString()` a `parseNextMachine()`, kde slouží pro zpracování případného řetězce určujícího číslo pásky TS, ke které se vztahuje daný aktuálně zpracovávaný symbol na hraně (případ `parseNextString()`) nebo stroj v uzlu (případ `parseNextMachine()`). `parseTapeNum()` je v uvedených metodách volána vždy v místě zpracovávání řetězce, kde se může vyskytovat informace o číslu pásky. Úkolem této metody je pak zjistit, zda se na daném místě číslo pásky skutečně nachází a pokud ano, tak jej zpracovat a vrátit v předem definovaném formátu. V případě, že se na daném místě řetězce informace o číslu pásky nevyskytuje, metoda žádnou část předaného řetězce nezpracuje a pouze vrátí výchozí hodnotu čísla pásky (1. páska).

8.2.3.4 Metoda `parseNextString()`

Metoda pro zpracování části řetězce zapsaného u hrany kompozitního diagramu. Konkrétně se zpracování řetězců s využitím této metody provádí tak, že je řetězec nejdříve rozdělen na podřetězce, jejichž oddělovačem je znak čárky. Následně jsou jednotlivé podřetězce vkládány na vstup této metody. Jejím úkolem pak je zpracovat předložený řetězec do předdefinovaného formátu s využitím metody `parseTapeNum()` pro zpracování čísel pásek.

8.2.3.5 Metoda `parseNextMachine()`

Metoda `parseNextMachine()` má stejný účel jako `parseNextString()`, ale pracuje s řetězci zapsanými uvnitř uzlů kompozitního diagramu. Práce s touto metodou se ale také liší, protože zpracovává řetězec na vstupu po jednotlivých základních stavebních blocích TS. Tento přístup je zde použit z důvodu, že se stejným způsobem provádí simulace jednoho kroku výpočtu Turingova stroje – vždy se provede simulace jednoho stavebního bloku TS. Metoda také slouží pro nastavování hodnot proměnných, na základě kterých se v GUI aplikace červeně obarvují jednotlivé právě odsimulované části kompozitního diagramu (např. proměnná `stateBB`).

8.2.3.6 Metoda `trivStep()`

Metoda `trivStep()` provádí simulaci jednoho kroku výpočtu Turingova stroje, přičemž je určena pro simulaci kroků v rámci jednoho uzlu kompozitního diagramu. Neprovádí tedy přechody mezi uzly, ale pouze simuluje spuštění následujícího stavebního bloku TS (že nějaký existuje, musí být zajištěno volající metodou). `trivStep()` využívá ke své činnosti tyto významné metody:

- `parseNextMachine()` – Pro zjištění, jaký stavební blok TS má být simulován.
- `runLR_Machine()` a `runLR_NMachine()` – Pro simulaci strojů typu L_a a $L_{\neg a}$.

- `moveHead()` – Pro simulaci strojů L nebo R.
- `setTapeSymbol()` – Pro simulaci zápisu symbolu na políčko pásky pod hlavou stroje.

Simulace provedení dalších podporovaných stavebních bloků TS ($S_L, D, \ldots$) jsou implementovány přímo v metodě `trivStep()`.

8.2.3.7 Metody `runLR_Machine()` a `runLR_NMachine()`

Jde o jednoduché pomocné metody, které za pomoci výše uvedeného `moveHead()` provádí posuv hlavy na vybrané pásce až na daný symbol (resp. až na jiný než daný symbol). V obou metodách je implementována jednoduchá detekce zacyklení TS, která se projeví v případě, že se hlava stroje posune na políčko, které dosud v simulaci ještě nikdy nebylo navštíveno a ani na něm nebyl zapsán vstupní řetězec, a současně je úkolem:

- metody `runLR_Machine()` posunout hlavu stroje doprava až na nějaký symbol X, kde X je libovolný symbol mimo blank (např. $R_a, R_b, \ldots$), nebo
- metody `runLR_NMachine()` posunout hlavu stroje doprava až na jiný symbol než blank ($R_{\neg\Delta}$).

V případě, že ke zmíněné situaci během simulace dojde, je vypsán dialog o zacyklení TS a simulace je ukončena.

8.2.3.8 Metoda `setTapeSymbol()`

Jednoduchá metoda pro zápis daného symbolu na vybranou pásku, konkrétně na políčko nad kterým se aktuálně nachází odpovídající hlava stroje.

8.2.3.9 Metoda `useThisEdge()`

Metoda `useThisEdge()` slouží pro provedení přechodu v kompozitním diagramu s využitím dané hrany. Po provedení přechodu se dále provádí simulace základního stavebního bloku TS, který se nachází na prvním místě v cílovém uzlu. Pro tento účel je zde použita metoda `trivStep()`. V metodě `useThisEdge()` také dochází k nastavování hodnot proměnných, na základě kterých se v GUI aplikace červeně obarvují hrany kompozitního diagramu (pole `lastUsedEdges`). Metoda také řeší případ, kdy hrana vede do speciálního uzlu parametrové konvence a je tedy třeba provést nastavení hodnot patřičným proměnným parametrové konvence. Před koncem metody je pak ještě volána metoda `prepareForLeavingVertex()` (více viz kapitola 8.2.3.10).

8.2.3.10 Metoda `prepareForLeavingVertex()`

Zavolání této metody má význam pouze v případech, kdy již byla provedena simulace posledního stavebního bloku TS v aktuálním uzlu kompozitního diagramu, v ostatních případech metoda neprovádí žádnou činnost. Pokud ovšem platí první možnost, metoda provádí plnění pole `feasibleOutGoingEdges`, které shromažďuje všechny hrany, přes které bude možné v dalším kroku simulace provést přechod do jiného uzlu kompozitního diagramu (využívá k tomu mimo jiné metodu `isFeasible()`). Zmíněné pole má význam zejména v situaci, kdy bude takových hran více. V tom případě se v GUI aplikace provádí

obarvení těchto hran modrou barvou a uživatel na ně může kliknout, aby vynutil provedení přechodu přes určitou hranu. Nutnost mít k dispozici metodu pro aktualizaci pole `feasibleOutGoingEdges` vychází z toho, že uživatel může během simulace měnit obsah pásek a tudíž se může změnit i množina hran, přes které je možné v následujícím kroku provést přechod.

8.2.3.11 Metoda `resetAllPreviousConfigurations()`

Uvedená metoda je první ze dvou metod, které implementují detekci cyklení Turingova stroje.

Než budeme pokračovat v popisu metody, nejprve si uveďme, jakým způsobem funguje implementovaná detekce cyklení TS. Aplikace detekuje cyklení TS na základě velmi jednoduchého principu, z čehož vyplývá, že může dojít k situaci, že TS bude cyklit, ale aplikace cyklení nebude detekovat. Principem využívaným v aplikaci konkrétně je sledovat konfigurace, do kterých se TS během výpočtu dostal a v případě, že navštíví některou konfiguraci podruhé, detekujeme cyklení. Speciálním případem je, pokud je TS nedeterministický. V případě, kdy TS na právě detekovaném cyklu měl možnost nedeterministicky zvolit následující přechod, je možné, že TS příště zvolí jiný přechod a tedy opustí tento cyklus. Pro rozlišení uvedených dvou případů (tedy případu, kdy na detekovaném cyklu je resp. není možné provést jiný než již provedený přechod) je u každé zaznamenané konfigurace uloženo číslo, které určuje počet dosud provedených nedeterministicky zvolených přechodů. Potom v případě, kdy zjistíme, že TS se vrátil do konfigurace, ve které se již jednou nacházel, stačí porovnat zmíněná čísla uvedená u těchto konfigurací. Pokud se shodují, nebyl na cyklu proveden žádný nedeterministický přechod a tedy je jisté, že se cyklení bude opakovat donekonečna (v tom případě je simulace ukončena). Pokud ovšem jsou čísla rozdílná, byl na cyklu proveden alespoň jeden nedeterministický přechod a tedy není jisté, zda bude TS v cyklení pokračovat (v tomto případě je uživatel na tuto skutečnost pouze upozorněn, ale simulace může pokračovat).

Vraťme se k popisu metody `resetAllPreviousConfigurations()`. Tato metoda provádí pouze smazání obsahu proměnné `AllPreviousConfigurations`, která obsahuje pro každý uzel kompozitního diagramu seznam již dosažených konfigurací (které obsahují tento uzel) s číslem specifikujícím, kolik bylo před dosažením této konfigurace provedeno nedeterministicky zvolených přechodů.

8.2.3.12 Metoda `addToAllPreviousConfigurationsAndCheck()`

Tato metoda je druhou z metod, které implementují detekci cyklení Turingova stroje (pro bližší popis mechanismu detekce cyklení viz kapitola 8.2.3.11), konkrétně řeší přidání aktuální konfigurace TS do seznamu již dosažených konfigurací a zároveň kontroluje, zda TS do stejné konfigurace nedospěl už dříve. Do zmíněného seznamu nejsou vkládány přímo konfigurace TS, ale pouze jejich hashe, dojde tím ke snížení paměťové náročnosti aplikace a také porovnávání krátkých hashů je rychlejší než porovnávání celých konfigurací, které mohou obsahovat dlouhé řetězce s obsahy pásek.

8.2.3.13 Metoda `step()`

Jde o nejdůležitější metodu třídy `TuringCore`, provádí simulaci jednoho kroku výpočtu zadaného Turingova stroje z aktuální konfigurace, přičemž využívá výše popsaných metod,

které realizují její jednotlivé části. Na nejvyšší úrovni metoda rozlišuje dva případy, které mohou nastat:

1. V předchozím kroku byla provedena simulace posledního základního stavebního bloku TS v aktuálním uzlu a nyní je třeba provést přechod do jiného uzlu s využitím hran kompozitního diagramu.
2. Pro simulaci dalšího kroku výpočtu stačí provést simulaci následujícího stavebního bloku TS v aktuálním uzlu. Tento případ je jednodušší a pro vlastní simulaci stačí využít metodu `trivStep()`.

První případ je složitější, popišme si jej proto nyní detailněji. Zde metoda `step()` využívá pole `feasibleOutGoingEdges` (více viz kapitola 8.2.3.10), kde na základě počtu hran v tomto poli provede následující:

- Pokud je počet hran v poli roven nule, pak ze zavedené sémantiky kompozitních diagramů Turingův stroj okamžitě přijme.
- Pokud je počet hran v poli roven jedné, pak je provedena simulace přechodu přes tuto hranu (metoda `useThisEdge()`).
- Pokud je hran v poli více, pak je následující hrana, přes kterou se provede přechod, vybrána podle následujících pravidel, přičemž se při jejich vyhodnocování postupuje od začátku a to, které bude jako první splněno, určuje vybranou hranu:
  - Pokud uživatel v GUI aplikace kliknul na některou z modře obarvených hran, pak je vybrána tato hrana.
  - Pokud je pro aktuální uzel stanovena nějaká preferovaná hrana a současně je možné přes tuto hranu provést přechod, pak je vybrána tato preferovaná hrana.
  - Jinak je hrana vybrána náhodně.

Po provedení simulace kroku výpočtu Turingova stroje je na závěr ještě spuštěna metoda `addToAllPreviousConfigurationsAndCheck()` (jen v případě, že je aktivována detekce cyklení TS).

K návratu z metody `step()` může dojít buď vyvoláním výjimky nebo klasickým návratem. Obecně lze říci, že výjimka je touto metodou vyvolána v případech, kdy je jisté, že se v simulaci nebude dále pokračovat. Patří sem tyto možnosti:

- Turingův stroj přijal svůj vstup.
- Hlava na některé pásce „přepadla“ vlevo.
- Došlo na simulaci speciálního stroje „D“ – TS okamžitě zamítne svůj vstup.
- Bylo detekováno cyklení TS po spuštění stroje typu $R_a, R_{\neg\Delta}, \ldots$ (nemusí být aktivní detekce cyklení TS – jde o mechanismus, který má zabránit zacyklení aplikace). Více viz kapitola 8.2.3.7.
- Bylo detekováno cyklení TS, protože došlo k návratu do již navštívené konfigurace (musí být aktivní detekce cyklení TS).

Naopak klasický návrat z metody `step()` signalizuje, že simulace může v budoucnu ještě pokračovat. Zde se rozlišují pouze dvě možnosti. Konkrétně zda bylo nebo nebylo detekováno cyklení TS v cyklu, ze kterého je možné se vymanit (jde o případ, kdy v některém uzlu na detekovaném cyklu bylo nedeterministicky rozhodnuto jít určitou cestou).

8.2.4 Třída `HaltingDetectionObj`

Jak už bylo zmíněno výše, třída `HaltingDetectionObj` implementuje heuristiku, jejíž účelem je snaha detekovat na základě analýzy kompozitního diagramu Turingova stroje, zda tento stroj vždy zastaví. Třída využívá princip popsaný v kapitole 5.6, který si nyní v krátkosti připomeneme.

Třída `HaltingDetectionObj` o aktuálním TS zahlásí, že jeho výpočet na všech vstupních řetězcích vždy skončí, pokud je výsledkem analýzy Turingova stroje zjištění, že po jeho spuštění bude v průběhu výpočtu docházet k opakovaným změnám v konfiguraci TS, kde součástí každé změny bude „zmenšení“ (podle předem definovaného fundovaného uspořádání) některé z položek nové konfigurace oproti odpovídající položce v původní konfiguraci. Protože z definice fundovaného uspořádání nemůže k takovému zmenšování docházet donekonečna, je jisté, že TS někdy v budoucnu zastaví.

Nyní už přejdeme k formálnějšímu popisu použitého principu a využívaných uspořádání.

8.2.4.1 Používaná uspořádání

Třída `HaltingDetectionObj` používá konkrétně následující uspořádání:

- $>_S$ – Uspořádání symbolů páskové abecedy TS, je vytvářeno ještě před započetím vlastní analýzy na základě průchodu celým kompozitním diagramem TS. Během průchodu hledáme místo, kde je v TS proveden přepis symbolu a_1 na symbol a_2. Pokud jsou symboly a_1 a a_2 shodné, pak mezi nimi relaci $>_S$ nezavádíme (ireflexivita relace $>_S$). Jinak pokud symboly a_1 a a_2 nejsou zatím v relaci, zavedeme mezi nimi novou relaci $a_1 >_S a_2$[5], jinak neprovádíme žádné změny (tímto dosáhneme asymetričnosti relace). Ačkoliv v samotné reprezentaci ve třídě `HaltingDetectionObj` není relace $>_S$ udržována jako tranzitivní (nejsou zde uvedeny patřičné relace mezi symboly), pracuje se s touto relací jako s tranzitivní (konkrétně je tranzitivita implementována pomocí rekurzivní metody `isGreaterThan()`). Tímto jsme splnili všechny vlastnosti, které musí mít binární relace, abychom ji mohli nazvat ostré uspořádání. Že se jedná navíc také o fundované uspořádání, není těžké nahlédnout, protože se v uspořádání nevyskytují cykly a množina symbolů, nad kterou je uspořádání definováno, je vždy konečná.

- $>_{KS}$ – Uspořádání na množině konfigurací TS (slouží k definici uspořádání $>_K$ – viz dále). Definice uspořádání $>_{KS}$ je následující.

  Definice 8.1. Nechť $M_H = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ je TS a K_1, K_2 jsou konfigurace TS M_H, kde $\forall i \in \{1, 2\} : K_i = (q_i, s_i, n_i), s_i = \Delta w_i \Delta^\omega = s_0^i s_1^i s_2^i \ldots$. Potom:

  $$K_1 >_{KS} K_2 \Leftrightarrow (\forall j : (s_j^1 >_S s_j^2 \vee s_j^1 = s_j^2)) \wedge (\exists k : s_k^1 >_S s_k^2)$$

  Dále také platí:

  $$K_1 =_{KS} K_2 \Leftrightarrow \forall j : s_j^1 = s_j^2$$

  V některých případech, pokud není jisté, zda bude při přechodu mezi konfiguracemi nějaký symbol nahrazen jiným, nastává problém, že uspořádání mezi těmito konfiguracemi může nabývat více možností (které si případně mohou navzájem odporovat).

[5] Jedinou výjimkou je, pokud je a_1 symbol blank. V tom případě je zavedena opačná relace ($a_2 >_S a_1$), aby se zajistilo, že blank bude v relaci $>_S$ vždy nejmenším prvkem a tedy pokud bude v TS docházet k jeho přepisu na jiné symboly, automaticky toto dále vyhodnotíme jako nechtěnou vlastnost.

V takovém případě zavádíme následující zjednodušení zápisu:

$$(K_1 >_{KS} K_2 \vee K_1 =_{KS} K_2) \Leftrightarrow (K_1 \geq_{KS} K_2)$$

Ve všech ostatních nepopsaných případech definujeme $(K_1 \,?_{KS}\, K_2)$.

Uvedené uspořádání $>_{KS}$ je fundované, protože je jen triviální nástavbou nad fundovaným uspořádáním $>_S$. Zde je důležité si uvědomit důvod zavedení výjimky popsané v poznámce u uspořádání $>_S$. Pokud bychom zmíněnou výjimku nezavedli a vždy ustavili relaci $a_1 >_S a_2$, mohlo by dojít k tomu, že by uspořádání $>_{KS}$ obecně nemuselo být fundované. Důvodem je, že v případě, že by v uspořádání $>_S$ existoval nějaký symbol x pro který by platilo $\Delta >_S x$, pak by v uspořádání $>_{KS}$ existovala nekonečná klesající posloupnost konfigurací $\Delta^\omega >_{KS} x\Delta^\omega >_{KS} xx\Delta^\omega >_{KS} \ldots$, což plyne z toho, že na pásce TS je k dispozici nekonečně mnoho symbolů Δ.

S tímto uspořádáním se ve třídě `HaltingDetectionObj` pracuje ve formě čísla x_1, kde $x_1 \in \{-2, -1, 0, 1, 2, 3\}$. Přiřazení hodnot čísla x_1 odpovídajícímu uspořádání je následující:

- -2 – $K_1 >_{KS} K_2$
- -1 – $K_1 \geq_{KS} K_2$
- 0 – $K_1 =_{KS} K_2$
- 1 – $K_2 \geq_{KS} K_1$
- 2 – $K_2 >_{KS} K_1$
- 3 – $K_1 \,?_{KS}\, K_2$

Jde tak o ekvivalentní reprezentaci uvedeného uspořádání.

- $>$ – Uspořádání pozic hlav TS. Protože jsou pozice hlav TS čísla z množiny $\mathbb{N}_0$, jde vlastně o klasické uspořádání „je větší než“ definované na této množině. Z kapitoly 5.6 pak víme, že jde o fundované uspořádání. S tímto uspořádáním se ve třídě `HaltingDetectionObj` přímo nepracuje, ale je důležité pro následující uspořádání.

- $>_1, >_2, >_3$ – Uspořádání na množině konfigurací TS (slouží k definici uspořádání $>_K$ – viz dále). Definice jednotlivých uspořádání je následující.

  Definice 8.2. Nechť $M_H = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ je TS a K_1, K_2 jsou konfigurace TS M_H, kde $\forall i \in \{1, 2\} : K_i = (q_i, \Delta w_i \Delta^\omega, n_i)$. Potom:

  - $K_1 >_1 K_2 \Leftrightarrow n_1 > n_2$,
  - $K_1 >_2 K_2 \Leftrightarrow n_1 - 1 > n_2$,
  - $K_1 >_3 K_2 \Leftrightarrow n_1 - 2 > n_2$.

  Dále také platí:

  - $K_1 =_1 K_2 \Leftrightarrow n_1 = n_2$,
  - $K_1 =_2 K_2 \Leftrightarrow |n_1 - n_2| \leq 1$,
  - $K_1 =_3 K_2 \Leftrightarrow |n_1 - n_2| \leq 2$.

V některých případech, pokud je posuv hlav při přechodu mezi konfiguracemi dán jistým intervalem, nastává problém, že pro určité uspořádání může nastat více možností (které si případně mohou navzájem odporovat). V takovém případě zavádíme následující zjednodušení zápisu:

$$\forall i \in \{1,2,3\} : (K_1 >_i K_2 \vee K_1 =_i K_2) \Leftrightarrow (K_1 \geq_i K_2)$$
$$\forall i \in \{1,2,3\} : (K_1 >_i K_2) \vee (K_2 >_i K_1) \Leftrightarrow (K_1 ?_i K_2)$$

Všechna uvedená uspořádání $>_1, >_2, >_3$ jsou také fundovaná, protože jsou opět jen triviální nástavbou nad fundovaným uspořádáním ($>$).

S těmito uspořádáními se ve třídě `HaltingDetectionObj` již pracuje, ale jsou zde reprezentovány ve formě dvojice čísel (x_2, x_3), kde $x_2, x_3 \in \{-3,-2,-1,0,1,2,3\}$. Číslo x_2 specifikuje levé (x_3 pak pravé) ohraničení intervalu, který určuje, jakých hodnot může nabývat rozdíl $n_2 - n_1$ při přechodu z konfigurace K_1 do K_2 (přičemž hodnoty -3 resp. 3 mají význam „posuv o 3 a více vlevo“ resp. „posuv o 3 a více vpravo“). Jde tak o ekvivalentní reprezentaci všech tří uspořádání dohromady, se kterou se lépe pracuje. Uveďme si dva příklady použití obou přístupů.

Příklad 8.1. *Nechť při výpočtu v rámci určité části kompozitního diagramu přejde TS M_H z konfigurace K_1 do K_2 a nechť se hlava stroje v rámci tohoto výpočtu posune buď doleva o libovolný počet políček nebo maximálně o jedno políčko doprava. Potom jsou uspořádání konfigurací K_1 a K_2 stanovena takto:*

- $K_1 ?_1 K_2$ – *Platí buď $K_1 >_1 K_2$ (pro případ, kdy se hlava posune o jedno a více políček doleva) nebo $K_2 >_1 K_1$ (případ, kdy se hlava posune o jedno políčko doprava) a nebo $K_1 =_1 K_2$ (případ, kdy hlava TS zůstane na místě).*
- $K_1 \geq_2 K_2$
- $K_1 \geq_3 K_2$

Popis stejného případu pomocí dvojice čísel (x_2, x_3) je pak následující: $(-3, 1)$ (-3 specifikuje, že se hlava stroje může posunout o libovolný počet políček vlevo a 1 určuje, že směrem vpravo se může posunout maximálně o jedno políčko pásky).

Příklad 8.2. *Nechť při výpočtu v rámci určité části kompozitního diagramu přejde TS M_H z konfigurace K_1 do K_2 a nechť se hlava stroje v rámci tohoto výpočtu posune buď o jedno nebo dvě políčka doprava. Potom jsou uspořádání konfigurací K_1 a K_2 stanovena takto:*

- $K_2 >_1 K_1$
- $K_2 \geq_2 K_1$ – *Platí buď $K_2 >_2 K_1$ (pro případ, kdy se hlava posune o dvě políčka doprava) nebo $K_1 =_2 K_2$ (případ, kdy se hlava posune o jedno políčko doprava).*
- $K_1 =_3 K_2$

Popis stejného případu pomocí dvojice čísel (x_2, x_3) je pak následující: $(1, 2)$.

- $>_K$ – Nejdůležitější uspořádání definované na množině konfigurací TS, které slučuje výše uvedená uspořádání $>_{KS}$ s trojicí uspořádání $>_1, >_2, >_3$ a používá se ve třídě `HaltingDetectionObj` pro rozhodnutí, zda daný TS vždy zastaví, nebo nejsme schopni odpovědět. Definice uspořádání $>_K$ je následující.

Definice 8.3. Nechť $M_H = (Q, \Sigma, \Gamma, \delta, q_0, q_F)$ je TS a K_1, K_2 jsou konfigurace TS M_H, kde $\forall i \in \{1, 2\} : K_i = (q_i, \Delta w_i \Delta^\omega, n_i)$. Potom:

$$K_1 >_K K_2 \Leftrightarrow K_1 >_{KS} K_2 \vee K_1 >_3 K_2$$

Ve všech ostatních nepopsaných případech definujeme $(K_1 \,?_K\, K_2)$.

Protože uspořádání $>_K$ jen využívá fundovaná uspořádání $>_{KS}$ a $>_3$, je samo fundované a ve třídě `HaltingDetectionObj` je vyjádřeno jako trojice čísel (x_1, x_2, x_3) (významy a možné hodnoty jednotlivých položek jsou vysvětleny u odpovídajících uspořádání výše).

Na závěr popisu používaných uspořádání si uveďme příklad, jak se v aplikaci provádí např. skládání dvou uspořádání $>_K$ „za sebe“ (viz popis metody `composeAllRelationsSeq()` v kapitole 8.2.4.4), následně si ještě uvedeme příklad skládání dvou uspořádání $>_K$ „vedle sebe“ (vztah nebo – viz popis metody `composeAllRelationsOr()` v kapitole 8.2.4.4).

Příklad 8.3. *Nechť při výpočtu v rámci určité části kompozitního diagramu přejde TS M_H vždy nejprve z konfigurace K_1 do K_2 a nechť uspořádání $>_K$ mezi těmito konfiguracemi je specifikováno trojicí $(-2, -3, -1)$, následně nechť TS M_H vždy přejde z konfigurace K_2 do K_3 a nechť uspořádání $>_K$ je specifikováno trojicí $(1, 1, 2)$. Potom je výsledné uspořádání mezi konfiguracemi K_1 a K_3 vzniklé jako složení uvedených uspořádání za sebe popsané trojicí $(3, -3, 1)$. Číslo 3 zde odpovídá tomu, že $K_1 \,?_{KS}\, K_3$ (protože skládáme za sebe $K_1 >_{KS} K_2$ a $K_3 \geq_{KS} K_2$). Následující položka v trojici (-3) odpovídá tomu, že směrem vlevo není posuv hlavy v rámci přechodu z K_1 do K_2 omezen a tudíž není omezen ani při přechodu z K_1 do K_3. Konečně, protože při přechodu z K_1 do K_2 se hlava TS posune vlevo vždy nejméně o jedno políčko pásky (-1 na poslední pozici v první trojici) a při přechodu z K_2 do K_3 se hlava TS posune vpravo maximálně o dvě políčka (2 na poslední pozici v druhé trojici), výsledek je, že po přechodu z K_1 do K_3 se hlava TS posune doprava maximálně o jedno políčko (1 na poslední pozici ve výsledné trojici).*

Příklad 8.4. *Nechť při výpočtu v rámci určité části kompozitního diagramu může TS M_H přejít z konfigurace K_1 do K_2 pomocí dvou různých přechodů a nechť uspořádání $>_K$ mezi těmito konfiguracemi je specifikováno trojicí $(-2, -3, -1)$ pro případ využití prvního přechodu a $(1, 1, 2)$ pro případ využití druhého přechodu. Potom je výsledné uspořádání mezi konfiguracemi K_1 a K_2 vzniklé jako složení uvedených uspořádání vedle sebe popsané trojicí $(3, -3, 2)$. Číslo 3 zde odpovídá tomu, že $K_1 \,?_{KS}\, K_2$ (protože může dojít k tomu, že $K_1 >_{KS} K_2$ nebo $K_2 \geq_{KS} K_1$). Následující položka v trojici (-3) odpovídá tomu, že směrem vlevo není posuv hlavy v rámci jednoho z možných přechodů omezen a tudíž není omezen ani ve výsledném uspořádání, které musí uvažovat obě možnosti přechodu. Podobná situace je i u poslední položky ve výsledné trojici. Zde je ale výsledné číslo 2 naopak převzato z trojice popisující druhý z možných přechodů, protože při využití tohoto přechodu se hlava TS posune doprava vždy maximálně o dvě políčka, což je ale více než při využití prvního přechodu.*

8.2.4.2 Popis činnosti třídy `HaltingDetectionObj`

Detekce zastavení pomocí třídy `HaltingDetectionObj` probíhá následovně:

- Před začátkem vlastní analýzy je za účelem odstranění případné kolize obarvení uzlů a hran kompozitního diagramu (viz obarvování dále) provedeno resetování simulace.

- Všechny uzly a hrany kompozitního diagramu TS jsou označeny jako neplatné. Následně jsou jako platné označeny pouze ty uzly a hrany, které jsou dostupné z uzlu, ze kterého začíná simulace.

- Následně je pro každý uzel kompozitního diagramu vypočtena trojice (x_1, x_2, x_3), která reprezentuje uspořádání $>_K$. Uspořádání $>_K$ vlastně popisuje pro každý uzel vztah mezi libovolnou konfigurací TS na vstupu uzlu (před provedením strojů obsažených v uzlu) a výslednou konfigurací na výstupu z uzlu. Uspořádání $>_K$ si lze také představit, že popisuje, jak bude modifikována konfigurace TS po „průchodu“ tímto uzlem.

- Vytvoření uspořádání $>_S$ (více viz u jeho popisu výše).

- V cyklu se snažíme celý kompozitní diagram „zjednodušit“ (viz kapitola 8.2.4.3) tak, aby výsledkem bylo uspořádání $>_K$ pro celý TS.

  – Pokud se podařilo celý TS zjednodušit do jediného uzlu, pak analýza končí a výsledkem je:

  $$K_1 >_K K_2 \Rightarrow \text{TS zastaví},$$

  kde K_1 je libovolná počáteční konfigurace a K_2 je odpovídající koncová konfigurace. Jelikož třída `HaltingDetectionObj` pracuje pouze s uspořádáním $>_K$, uvažuje tak všechny možné počáteční konfigurace najednou. Pro určení výsledku, zda TS zastaví, proto stačí na konci analýzy vyhodnotit výslednou trojici (x_1, x_2, x_3) a pokud je splněna uvedená podmínka, můžeme říci, že TS vždy zastaví. Pokud analýza zjistí, že TS vždy zastaví, je tento výsledek uživateli prezentován v GUI aplikace ve formě obarvení celého kompozitního diagramu oranžovou barvou.

  – Jinak nemůžeme o TS jako celku rozhodovat, zda zastaví či nikoliv. Ve třídě `HaltingDetectionObj` je speciálně pro tyto případy implementováno také obarvování na oranžovo i pouze těch části kompozitního diagramu TS, u kterých bylo zjištěno, že vždy zastaví (viz metoda `colorHaltingCell()`).

8.2.4.3 Technika zjednodušování kompozitního diagramu

Abychom byli schopni zjistit, zda daný TS zastaví, je třeba získat uspořádání $>_K$ popisující celý TS a to následně vyhodnotit. Výpočet uspořádání $>_K$ celého TS z jednotlivých uspořádání definovaných pro každý uzel kompozitního diagramu (viz kapitola 8.2.4.2) je ve třídě `HaltingDetectionObj` implementován pomocí tzv. techniky zjednodušování.

Princip zjednodušování kompozitního diagramu je velmi jednoduchý, postupujeme následovně:

- Každý dosud platný uzel[6] kompozitního diagramu je testován, zda vyhovuje některému z předdefinovaných „vzorů“ pro zjednodušování.

- Pokud nevyhovuje žádnému vzoru, pokračujeme dalším platným uzlem. Pokud žádný další platný uzel neexistuje, zjednodušování končí neúspěchem.

[6] Jak už bylo uvedeno dříve, na začátku jsou uzly a hrany označeny jako platné, pokud jsou dostupné z uzlu, ze kterého začíná simulace.

- Pokud uzel některému vzoru vyhovuje, dojde ke zjednodušení kompozitního diagramu v tom smyslu, že uspořádání $>_K$ z jiných uzlů (definovaných vzorem) sloučíme s tím z aktuálního uzlu a zpracované uzly a hrany nebudeme v dalším procesu zjednodušování vůbec uvažovat (zneplatníme je). Zjednodušování končí úspěchem.

Tento postup opakujeme do té doby, dokud zjednodušování kompozitního diagramu končí úspěchem. Jakmile dojde k tomu, že se během jedné iterace žádná část kompozitního diagramu nezjednoduší (neúspěch), pak jsme buď celý kompozitní diagram redukovali do jediného uzlu, nebo jsou některé jeho části natolik složité, že nevyhovují žádnému z definovaných vzorů.

8.2.4.4 Popis metod třídy `HaltingDetectionObj`

Hlavní principy použité v této třídě byly již popsány výše, nyní si jen stručně charakterizujme její významné metody.

- `isGreaterThan()` – Určuje, zda jsou dva předané symboly páskové abecedy v relaci $>_S$.
- `getVertexRelations()` – Vrací uspořádání $>_K$ platné pro daný základní stavební blok TS ve formě trojice (x_1, x_2, x_3) (např. pro stroj R vrací trojici $(0, 1, 1)$).
- `composeAllRelationsOr()` – Slučuje dvojici uspořádání $>_K$ do jediného. Ve výsledném uspořádání je reflektováno to, že dvě části kompozitního diagramu (které jsou popsány původními uspořádáními) mají mezi sebou vztah „nebo“, tedy že v případné simulaci by byla vždy provedena jen jedna z těchto částí.
- `composeAllRelationLoop()` – Provádí převod uspořádání $>_K$ na nové, které popisuje kompozitní diagram, který donekonečna opakuje provádění původní části kompozitního diagramu, která je popsána uspořádáním na vstupu této metody.
- `composeAllRelationsSeq()` – Slučuje dvojici uspořádání $>_K$ do jediného. Ve výsledném uspořádání je reflektováno to, že dvě části kompozitního diagramu (které jsou popsány původními uspořádáními) mají mezi sebou vztah „a“, tedy že v případné simulaci by byla vždy nejprve provedena první a následně druhá z těchto částí.
- `getIncomingSymbolsFromEdge()` – Vrací symboly vyskytující se na dané hraně zpracované v definovaném formátu.
- `setCellAsValid()` – Rekurzivní metoda pro nastavení daného uzlu jako platného včetně všech uzlů, které jsou z něho dostupné po odchozích hranách.
- `getLastSimplifiedCell()` – Provádí „přeskakování“ neplatných částí kompozitního diagramu. Konkrétně pro uzel na vstupu (označme jej u_1) vrátí první uzel (označme jej u_2), který je z u_1 dostupný (i pomocí neplatných hran) a který ještě má nějaké platné hrany (uspořádání $>_K$ z uzlu u_2 je ale již vloženo do uzlu u_1). Pokud se takový uzel u_2 nalezne, je naděje, že bude ještě možné provádět nějaké zjednodušování kompozitního diagramu.
- `simplifyTS()` – Provede jeden krok zjednodušení kompozitního diagramu (viz kapitola 8.2.4.3).

- `run()` – Hlavní metoda třídy `HaltingDetectionObj`, je spouštěna z třídy `GUIObj` při požadavku na spuštění detekce zastavení (viz kapitola 8.2.4.2).
- `colorHaltingCell()` – Metoda spouštěná po každém úspěšném zjednodušení kompozitního diagramu. Jejím účelem je obarvit v GUI aplikace oranžově ty části kompozitního diagramu, které vždy zastaví (na konci případně obarví celý kompozitní diagram).
- `halts()` – Vyhodnocuje na základě uspořádání $>_K$ předaného ve formě trojice (x_1, x_2, x_3), zda daná část kompozitního diagramu TS (popř. celý) zastaví nebo nejsme schopni odpovědět.

8.2.4.5 Rozšíření detekce zastavení na vícepáskové TS

Až do této chvíle jsme vždy uvažovali, že analyzovaný TS je jednopáskový. Nyní si v krátkosti popíšeme rozšíření detekce zastavení také na vícepáskové TS, které je implementováno ve třídě `HaltingDetectionObj`. Jde v zásadě jen o rozšíření všech dosud uvedených konceptů na více pásek. Výsledkem potom je, že hodnoty trojice (x_1, x_2, x_3) jsou udržovány zvlášť pro každou pásku TS a zvlášť se s nimi také pracuje.

Pokud budeme pro účely vícepáskových TS značit uspořádání $>_K$ vztahující se k i-té pásce jako $>_K^i$, pak na konci analýzy k-páskového TS můžeme říci, že:

$$\exists i \in \{1, \ldots, k\} : K_1 >_K^i K_2 \Rightarrow k\text{-páskový TS vždy zastaví,}$$

kde K_1 je libovolná počáteční konfigurace a K_2 je odpovídající koncová konfigurace.

8.2.5 Třída `TmxCellEditor`

Třída `TmxCellEditor` implementuje vytváření a rušení editoru, který se objeví v části okna aplikace, která slouží pro zakreslení kompozitního diagramu, v okamžiku, kdy je uživatelem iniciována editace uzlu nebo hrany. Účelem editoru je tedy vytvořit prostor, ve kterém bude možné editovat obsah vybraného uzlu nebo hrany. Tato třída ale pouze přepisuje některé metody původní třídy `mxCellEditor` z knihovny JGraphX. U dále uvedených metod bude tedy jen v krátkosti zmíněno, co bylo v dané metodě změněno nebo přidáno oproti její původní implementaci.

8.2.5.1 Metoda `stopEditing()`

Tato metoda je volána za účelem ukončení editace ať už uzlu nebo hrany grafu (kompozitního diagramu). Úpravy implementace této metody zahrnují hlavně přidání konverze řetězce zadaného uživatelem do podoby HTML s využitím metody `cStringToHTML()`. Pro více informací o prováděných konverzích mezi řetězci viz kapitola 8.2.2.13.

8.2.6 Třída `TmxGraph`

Třída `TmxGraph` popisuje strukturu grafu (kompozitního diagramu) a obsahuje metody pro manipulaci s ním, ale nezabývá se jeho vykreslováním. Tato třída ale pouze přepisuje některé metody původní třídy `mxGraph` z knihovny JGraphX. U dále uvedených metod bude tedy jen v krátkosti zmíněno, co bylo v dané metodě změněno nebo přidáno oproti její původní implementaci.

8.2.6.1 Metoda `isCellEditable()`

Metoda odpovídá na dotaz, zda je možné daný uzel nebo danou hranu editovat. Změna provedená v implementaci této metody zahrnuje pouze zakázání editace speciálních uzlů parametrové konvence.

8.2.6.2 Metoda `isCellResizable()`

Tato metoda určuje, zda má být možné měnit velikost daného uzlu nebo hrany. Nová implementace pouze zakazuje měnit velikost libovolného uzlu.

8.2.7 Třída `TmxGraphComponent`

Třída `TmxGraphComponent` zajišťuje vykreslení grafu (kompozitního diagramu), jehož reprezentace je uložena ve třídě `TmxGraph`. Tato třída ale opět pouze přepisuje některé metody původní třídy `mxGraphComponent` z knihovny JGraphX. U dále uvedených metod bude tedy jen v krátkosti zmíněno, co bylo v dané metodě změněno nebo přidáno oproti její původní implementaci.

8.2.7.1 Metoda `installDoubleClickHandler()`

Tato metoda vytváří posluchače, který ošetřuje některé případy, kdy uživatel klikne tlačítkem myši do prostoru s grafem. Konkrétně řeší situace, kdy má dojít k započetí a také k ukončení editace uzlu nebo hrany. Změna provedená v implementaci této metody zahrnuje pouze spuštění metody `ParConvSetVariables()` namísto standardní editace ve chvíli, kdy by mělo dojít k editaci hrany, která vychází ze speciálního uzlu parametrové konvence (více viz kapitola 8.2.2.18).

Kapitola 9

Závěr

V této práci byla představena základní teorie týkající se problematiky Turingových strojů a jejich variant včetně možných forem popisu těchto strojů. V dalších kapitolách pak lze nalézt také teorii týkající se problému zastavení a analýzy složitosti výpočtů Turingova stroje. Dále byla navržena a implementována aplikace, která umožňuje editaci Turingových strojů zapsaných pomocí kompozitních diagramů a také simulaci jejich výpočtu na zadané vstupní konfiguraci (simulaci je možné krokovat ručně, nechat běžet automaticky nebo tyto přístupy libovolně kombinovat). Aplikace podporuje práci jak s vícepáskovými tak i s nedeterministickými Turingovými stroji. V aplikaci je dále implementována jednoduchá detekce cyklení TS na principu detekce návratu TS do stejné konfigurace a také heuristika pro částečné řešení problému zastavení Turingova stroje (obě funkce podporují jak vícepáskové tak i nedeterministické TS). V rámci práce bylo také vytvořeno 42 příkladů Turingových strojů, které přijímají daný jazyk resp. vyčíslují určitou funkci, dále 3 příklady TS zaměřené na ilustraci funkce detekce cyklení TS implementované v aplikaci a také 5 příkladů TS demonstrujících funkčnost heuristiky pro řešení problému zastavení (popis jednotlivých příkladů je uveden v příloze B).

Ve výsledné aplikaci je kromě výše uvedené funkcionality implementována také možnost ukládat a načítat vytvořené kompozitní diagramy TS spolu s dalšími informacemi jako jsou obsahy pásek, pozice hlav, proměnné parametrové konvence včetně symbolů uložených v nich apod. Aplikace podporuje všechny standardně používané základní stavební bloky TS, navíc je podporován speciální stroj „D“. V kompozitních diagramech je samozřejmě možné používat parametrovou konvenci a formát řetězců uváděných u hran je shodný s formátem, ve kterém se tyto řetězce v kompozitních diagramech standardně zapisují (včetně symbolu negace). Pro zápis řetězců do uzlů a ke hranám kompozitních diagramů se používá speciální syntaxe, která usnadňuje zápis TS do aplikace a která je následně konvertována do podoby, která celkově zpřehledňuje vytvořený kompozitní diagram TS. Změny obsahu pásek je v aplikaci možné provádět buď po jednotlivých symbolech nebo je možné využít editace celé pásky najednou.

V aplikaci se za běhu simulace červeně obarvují ty části kompozitního diagramu, jejichž simulace byla dokončena v předchozím kroku, uživatel tak snadněji získá přehled o aktuálním stavu simulace. Aplikace dále i za běhu simulace podporuje provádění některých změn např. obsahů pásek, pozic hlav apod. U automatického i manuálního režimu simulace je možné zvolit počet kroků výpočtu TS, které se provedou najednou. U automatického režimu je pak navíc ještě možné zvolit rychlost provádění simulace. V případě, kdy se během simulace zjistí, že následující přechod lze provést přes více hran kompozitního diagramu (nedeterministické TS), je možné hranu, která se nakonec použije, zvolit buď přímo

v kompozitním diagramu kliknutím na některou z modře zbarvených hran, nebo se použije preferovaná hrana (v případě, že je ustanovena), jinak se hrana vybere náhodně. V rámci samotné aplikace je ke všem jejím částem dostupná podrobná nápověda s příklady použití, kterou je možné si nechat zobrazit z horního tlačítkového menu.

Funkce aplikace by šly samozřejmě dále rozšiřovat, jmenujme např. implementace dalších zjednodušujících vzorů v heuristice pro řešení problému zastavení nebo rozšíření detekce cyklení TS o další přístupy.

Aplikace může najít své využití zejména při výuce teoretické informatiky, kde může posloužit např. pro demonstraci činnosti daného Turingova stroje.

Literatura

[1] TURING, A. M. On Computable Numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society.* 1936, roč. 42. s. 230–265.

[2] HOPCROFT, J. E., MOTWANI, R. a ULLMAN, J. D. *Introduction to Automata Theory, Languages and Computation.* 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. 521 s. ISBN 0-201-44124-1.

[3] SIPSER, M. *Introduction to the Theory of Computation.* 2nd ed. Boston: Thomson, 2006. 456 s. ISBN 0-534-95097-3.

[4] SISKA, J. *Simulátor Turingových strojů popsaných pomocí kompozitních diagramů.* Brno: FIT VUT v Brně, 2014. Semestrální projekt.

[5] ČEŠKA, M., VOJNAR, T. a SMRČKA, A. *Teoretická informatika, Studijní opora.* Brno: FIT VUT v Brně, 2013. Dostupné na: <http://www.fit.vutbr.cz/study/courses/TIN/public/Texty/oporaTIN.pdf>.

[6] ČERNÁ, I., KŘETÍNSKÝ, M. a KUČERA, A. *Automaty a formální jazyky I.* Brno: FI MU, 2002.

[7] HOLUB, Š. *Složitost pro kryptografii.* Praha: Matematicko-fyzikální fakulta, Univerzita Karlova, 2012. Dostupné na: <http://www.karlin.mff.cuni.cz/%7Eholub/skripta/slozitost.pdf>.

[8] KOZEN, D. C. *Automata and Computability.* 1st ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1997. 400 s. ISBN 0-387-94907-0.

[9] KRESLÍKOVÁ, J. a MARTINEK, D. *Základy programování, Studijní opora.* Brno: FIT VUT v Brně, 2006.

[10] TVRDÝ, F. *Turingův test, Filosofické aspekty umělé inteligence.* Olomouc: Filosofická fakulta, Univerzita Palackého v Olomouci, 2011. Disertační práce.

[11] CHURCH, A. A Set of Postulates for the Foundation of Logic Part I. *Annals of Mathematics.* 1932, roč. 33, č. 2. s. 346–366.

[12] CHURCH, A. A Set of Postulates for the Foundation of Logic Part II. *Annals of Mathematics.* 1933, roč. 34, č. 2. s. 839–864.

[13] POST, E. L. Finite Combinatory Processes – Formulation 1. *The Journal of Symbolic Logic.* 1936, roč. 1, č. 3. s. 103–105.

[14] POST, E. L. Formal Reductions of the General Combinatorial Decision Problem. *American Journal of Mathematics*. 1943, roč. 65, č. 2. s. 197–215.

[15] GÖDEL, K. *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*. Hewlett, New York: Raven Press Books, 1965. On Undecidable Propositions of Formal Mathematical Systems, s. 5–38.

[16] ŠTĚPÁNEK, P. *Zastavování výpočtů*. 2007. Dostupné na: <http://ktiml.mff.cuni.cz/teaching/files/materials/LP12.pdf>.

[17] RICH, E. *Automata, Computability and Complexity: Theory and applications*. 1st ed. Upper Saddle River, New Jersey: Pearson Prentice Hall, 2008. 1099 s. ISBN 978-0-13-228806-4.

[18] ZULEGER, F., GULWANI, S., SINN, M. et al. Bound Analysis of Imperative Programs with the Size-change Abstraction. In *Proceedings of the 18th International Conference on Static Analysis*. Berlin, Heidelberg: Springer-Verlag, 2011. s. 280–297. Dostupné na: <http://research.microsoft.com/en-us/um/people/sumitg/pubs/sas11-bound.pdf>. ISBN 978-3-642-23701-0.

Seznam příloh

Příloha A

Uživatelská dokumentace k aplikaci

A.1 Instalace

K překladu i spuštění aplikace je nutné mít nainstalovánu aktuální verzi Java Runtime Environment (JRE), kterou lze nalézt na `http://java.com/en/`. Aplikace byla psána a testována na verzi JRE 7 Update 25 firmy Oracle. Pro bezproblémový překlad a funkčnost aplikace je doporučeno mít nainstalovanou stejnou verzi. Po nainstalování Javy je možné aplikaci spustit buď přímo pomocí dodaného jar archívu „TuringSimulator.jar“, nebo lze provést překlad aplikace znovu a spustit takto vytvořený jar archív.

A.1.1 Překlad

Pro překlad aplikace je třeba mít nainstalován Java Development Kit (JDK) ve stejné verzi jako výše zmíněný JRE a také nástroj ant, který je možné nalézt ke stažení na domovských stránkách projektu `http://ant.apache.org/`. Bez správně nainstalovaného nástroje ant nelze aplikaci automatizovaně přeložit. Pro zahájení překladu aplikace se nejprve přepneme do složky „Aplikace/Zdrojove_kody“ a zadáme v příkazovém řádku příkaz `ant`. Po překladu lze spustitelný jar archív aplikace nalézt ve složce „Aplikace/Zdrojove_kody/dist“, nebo jej lze spustit ze složky „Aplikace/Zdrojove_kody“ zadáním příkazu `ant run`.

A.2 Používání aplikace

Podrobný popis jednotlivých funkcí a režimů aplikace lze nalézt v nápovědě integrované v aplikaci. Nápovědu je možné zobrazit z horního tlačítkového menu, konkrétně kliknutím na tlačítko „Nápověda“ a vybráním požadované sekce nápovědy.

A.3 Testované systémy

Aplikace byla testována na operačních systémech Windows XP SP3, Windows 7 Professional SP1, testování základní funkčnosti aplikace proběhlo také na Ubuntu 12.04. S výjimkou OS Ubuntu (ve kterém je v aplikaci použito jiné písmo a tudíž mohou být některé texty v aplikaci špatně zarovnány) se aplikace na uvedených systémech chová, jak bylo původně zamýšleno. Provoz aplikace na ostatních kombinacích OS/verze Javy by měl být samozřejmě také možný, ale aplikace pro ně není odladěna.

Příloha B

Popis přiložených příkladů TS

V této příloze bude pro každý z příkladů TS, který je přiložen u aplikace, uveden jazyk, který daný TS přijímá resp. funkce, kterou vyčísluje.

1. Přijímá jazyk $L_1 = \{a^n b^m c^o \mid n, m, o \in \mathbb{N}_0\}$.
2. Vypočítá rozdíl dvou binárních čísel. Vstupní hodnoty čísel $x_1, x_2 \in \{0,1\}^+$ jsou očekávány na pásce ve formátu $\Delta x_1 \# x_2 \Delta\Delta \ldots$ a výsledkem bude $\Delta x \Delta\Delta \ldots$, kde:
   - $x \in \{0,1\}^*$ a $x = x_1 - x_2$, pokud $(x_1 - x_2) \geq 0$
   - Jinak $x = 0$
3. Dekrementuje binární číslo $x \in \{0,1\}^*$ na vstupu o jedničku. Pokud bylo $x = 0$, pak je výsledkem 0.
4. Přijímá jazyk $L_4 = \{a^i b^j c^k \mid i, j, k \in \mathbb{N}_0 \wedge j > i + k\}$.
5. Přijímá jazyk $L_5 = \{IHTIN\}$ (převzato ze zadání 4. příkladu 3. domácího úkolu do předmětu TIN z roku 2010).
6. Reprezentace tzv. busy beaver Turingova stroje v kompozitním diagramu. Zde jde o stroj se třemi stavy.
7. Opět reprezentace busy beaver Turingova stroje v kompozitním diagramu. Zde jde ale o stroj se čtyřmi stavy.
8. Přijímá jazyk $L_8 = \{w \mid w \in \{a,b,c,d\}^* \wedge \#_a(w) = \#_b(w) \wedge \#_c(w) = \#_d(w)\}$, kde $\#_x(w)$ je počet symbolů x v řetězci w.
9. Přijímá jazyk $L_9 = \{w_1 \# w_2 \mid w_1, w_2 \in \{a,b\}^* \wedge (\#_a(w_1) = \#_a(w_2) \vee \#_b(w_1) = \#_b(w_2))\}$.
10. Přijímá jazyk $L_{10} = \{1^n \mid n$ je číslo z Fibonacciho posloupnosti$\}$.
11. Přijímá jazyk $L_{11} = \{a^{kn} b^n \mid k, n \in \mathbb{N}\}$.
12. Převede řetězec $w_1 \in \{a,b,c\}^+$, kde $\#_a(w_1) = n_a$, $\#_b(w_1) = n_b$ a $\#_c(w_1) = n_c$ na řetězec $w_2 = a^{n_a} b^{n_b} c^{n_c}$.
13. Přijímá jazyk $L_{13} = \{ww^R \mid w \in \{a,b\}^*\}$.

14. Převádí binární číslo $x \in \{0,1\}^*$ ze vstupu na unární. Výsledek je na konci výpočtu uložen na druhé pásce.

15. Vypočítá největší společný dělitel (GCD) dvou čísel, která jsou zadána na prvních dvou páskách. Číslo n se zadává v podobě unárního kódu a^n. Výsledek je na konci výpočtu uložen na třetí pásce.

16. Přijímá jazyk $L_{16} = \{w \mid w \in \{a,b,c\}^* \wedge w$ obsahuje alespoň dva výskyty podřetězce $w', \text{kde } w' \in \{a,b,c\}^4\}$.

17. Přijímá jazyk $L_{17} = \{ww \mid w \in \{a,b\}^*\}$.

18. Přijímá jazyk $L_{18} = \{a^{2^n} \mid n \in \mathbb{N}_0\}$.

19. Konkatenuje 1–6 řetězců (přesný počet je dán číslem n zapsaným na vstupu pomocí číslice $1, \ldots, 6$). Očekává vstup ve tvaru $\Delta n \Delta^{m_1} w_1 \Delta \ldots \Delta^{m_n} w_n \Delta\Delta \ldots$, kde $1 \leq n \leq 6$ a $\forall i \in \{1, \ldots, n\} : m_i > 0 \wedge w_i \in \{a,b\}^+$ (m_i tedy popisuje počet symbolů Δ oddělujících řetězec w_i od předcházející položky na vstupu). Výsledek má pak tvar $\Delta w_1 \ldots w_n \Delta\Delta \ldots$.

20. Přijímá jazyk $L_{20} = \{a^n \mid n$ není prvočíslo$\}$.

21. Přijímá jazyk $L_{21} = \{a^n \mid n$ je prvočíslo$\}$.

22. Ke vstupnímu řetězci $w \in \{x,y\}^*$ přidá reverzovaný řetězec w^R. Výsledek má pak tvar $\Delta w \Delta w^R \Delta\Delta \ldots$.

23. Přijímá jazyk $L_{23} = \{w \mid w \in \{ab^*\}^+ \wedge$ pro všechny podřetězce w ve tvaru ab^iab^ja platí $i < j \wedge$ pro sufix w ve tvaru ab^iab^j platí $i < j\}$.

24. Vypočítá druhou mocninu čísla $n \in \mathbb{N}_0$, které se zadává v podobě unárního kódu a^n.

25. Vypočítá faktoriál čísla $n \in \mathbb{N}_0$, které se zadává v podobě unárního kódu a^n.

26. Vypočítá druhou odmocninu čísla $n \in \mathbb{N}_0$ zaokrouhlenou nahoru na nejbližší celé číslo. Číslo n se zadává v podobě unárního kódu a^n. Výsledek je na konci výpočtu uložen na druhé pásce.

27. Přijímá jazyk $L_{27} = \{a^n b^m c^k \mid n,m,k \in \mathbb{N}_0 \wedge m \leq n \wedge m \leq k\}$.

28. Přijímá jazyk $L_{28} = \{ww^Rw \mid w \in \{a,b\}^*\}$.

29. Přijímá jazyk $L_{29} = \{a^ib^j \mid i,j \in \mathbb{N}_0 \wedge i + j = 5\}$.

30. Přijímá jazyk $L_{30} = \{a^ib^j \mid i,j \in \mathbb{N}_0 \wedge i - j = 5\}$.

31. Přijímá jazyk $L_{31} = \{w \mid w \in \{Y,N\}^* \wedge \#_Y(w) \geq 2 \wedge \#_N(w) \leq 2\}$.

32. Přijímá jazyk $L_{32} = \{w \mid w \in \{a,b\}^* \wedge$ každý výskyt symbolu a je předcházen i následován alespoň jedním symbolem $b\}$.

33. Přijímá jazyk $L_{33} = \{w \mid w \in \{a,b\}^* \wedge w \notin \{a,b\}^*ba\}$.

34. Přijímá jazyk $L_{34} = \{w \mid w \in \{0,1\}^+ \wedge w$ je binární číslo dělitelné čtyřmi$\}$.

35. Přijímá jazyk $L_{35} = \{w \mid w \in \{a,b\}^* \wedge w$ obsahuje jako podřetězec aa i $bb\}$.

36. Přijímá jazyk $L_{36} = \{w \mid w \in \{0,1\}^* \wedge (|w| \text{ je sudé} \vee w \text{ začíná na } 11)\}$.

37. Ke vstupnímu řetězci $w \in \{a,b\}^*$ přidá řetězec 1^n, kde $n = \#_a(w)$. Výsledek má pak tvar $\Delta w 1^n \Delta\Delta \ldots$.

38. Vypočítá podíl čísla $n \in \mathbb{N}_0$ číslem tři zaokrouhlený dolů na nejbližší celé číslo. Číslo n se zadává v podobě unárního kódu a^n.

39. Ke vstupnímu řetězci $w \in \{0,1\}^*$ přidá řetězec $p \in \{0,1\}$, který je výsledkem výpočtu sudé parity w. Výsledek má pak tvar $\Delta w \Delta p \Delta\Delta \ldots$.

40. Převádí hexadecimální číslo $w \in \{0,1,\ldots,9,A,B,C,D,E,F\}^*$ na binární.

41. Očekává vstup ve tvaru $\Delta w_1 \Delta \ldots \Delta w_n \Delta\Delta \ldots$, kde $n \geq 0 \wedge (\forall i \in \{1,\ldots,n\} : w_i \in \{0,1\}^+)$. Definujme operaci logického součinu aplikovanou na dva řetězce $x_1, x_2 \in \{0,1\}^+$ (jejíž výsledkem je řetězec $x_0 \in \{0,1\}^+$) následovně (nechť $\forall i \in \{0,1,2\} : x_i = x_0^i \ldots x_{l_i-1}^i$, kde $l_i = |x_i|$):

    - Pokud $|x_1| = |x_2|$, pak $|x_0| = |x_1|$ $(= |x_2|)$ a $\forall i \in \{0,\ldots,l_0-1\} : x_i^0 = x_i^1 \,\&\, x_i^2$, kde & je standardní bitová operace logického součinu.
    - Pokud $|x_1| < |x_2|$, pak vytvoříme nový řetězec $x_1' = 0^j x_1$, kde $j = |x_2| - |x_1|$. Nyní platí $|x_1'| = |x_2|$, výsledek operace pak určíme podle předchozího bodu. Analogicky postupujeme pro případ $|x_1| > |x_2|$.

    Po zastavení TS bude páska ve tvaru $\Delta w \Delta\Delta \ldots$, kde w je výsledek výše definované operace logický součin aplikované postupně na všechny w_i.

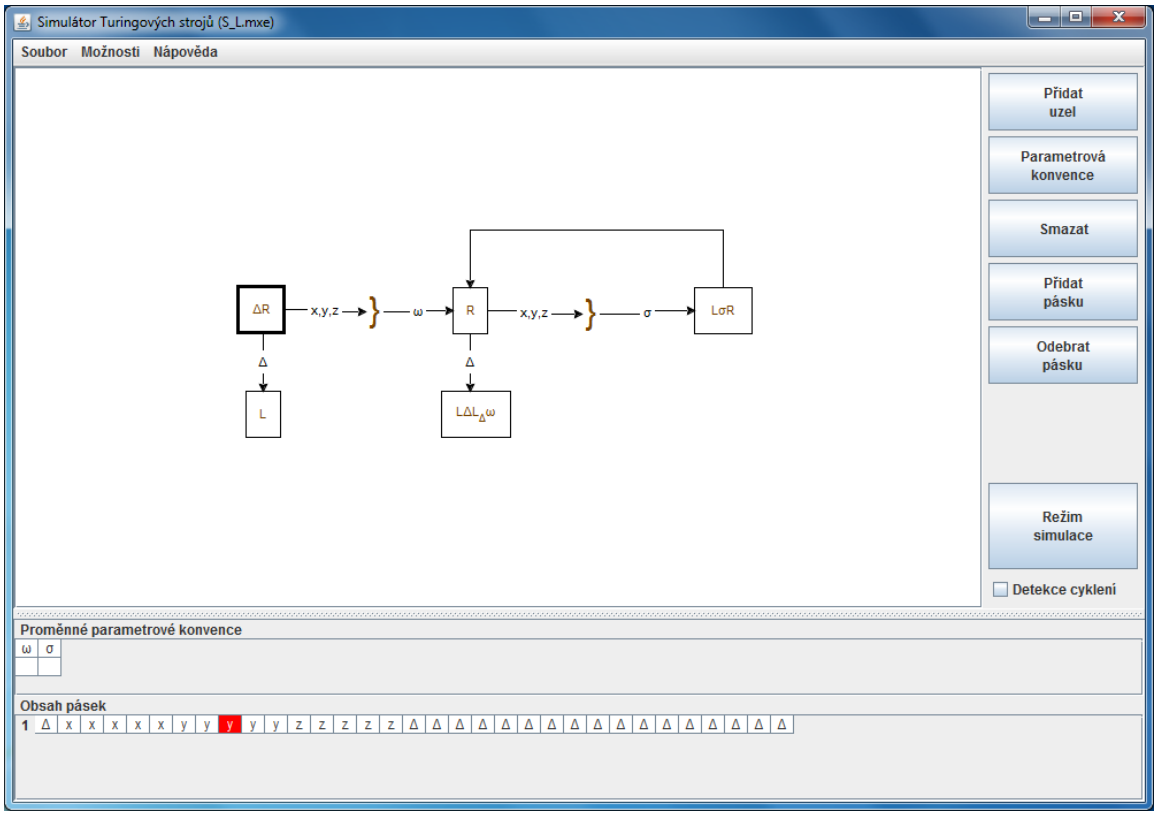
42. Očekává vstup ve tvaru $\Delta w_0 \Delta w \Delta\Delta \ldots$, kde $w_0 \in \{d,e\}, w \in \{a,b,c,\ldots,z\}^*$. Výsledek má pak tvar $\Delta w_0 \Delta w' \Delta\Delta \ldots$, kde w' je buď zašifrovaná verze (případ $w_0 = e$) nebo dešifrovaná verze (případ $w_0 = d$) řetězce w. Pro šifrování je použita Caesarova šifra.

43–45. Tyto Turingovy stroje neprovádí „užitečnou“ činnost, slouží pouze pro ilustraci funkce detekce cyklení TS implementované v aplikaci.

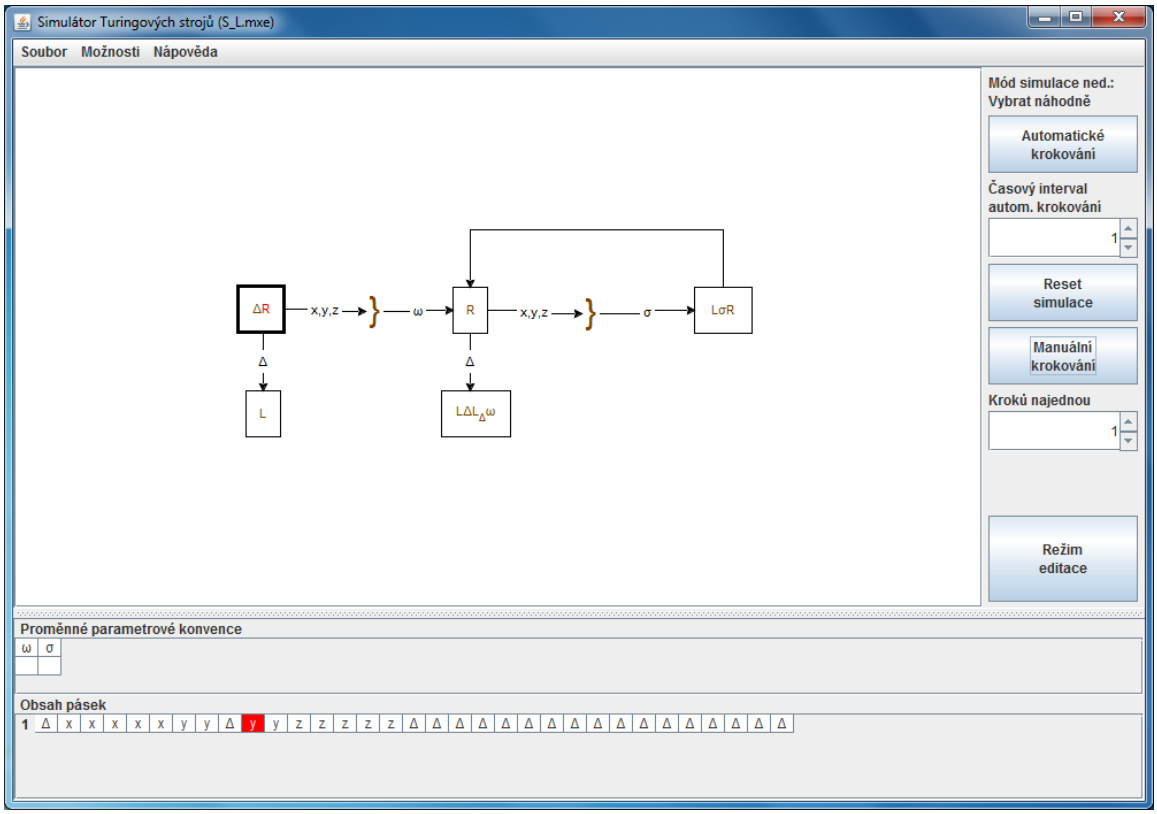
46–50. Tyto Turingovy stroje také neprovádí nic užitečného, jsou určeny pro demonstraci funkčnosti heuristiky pro řešení problému zastavení.

Příloha C

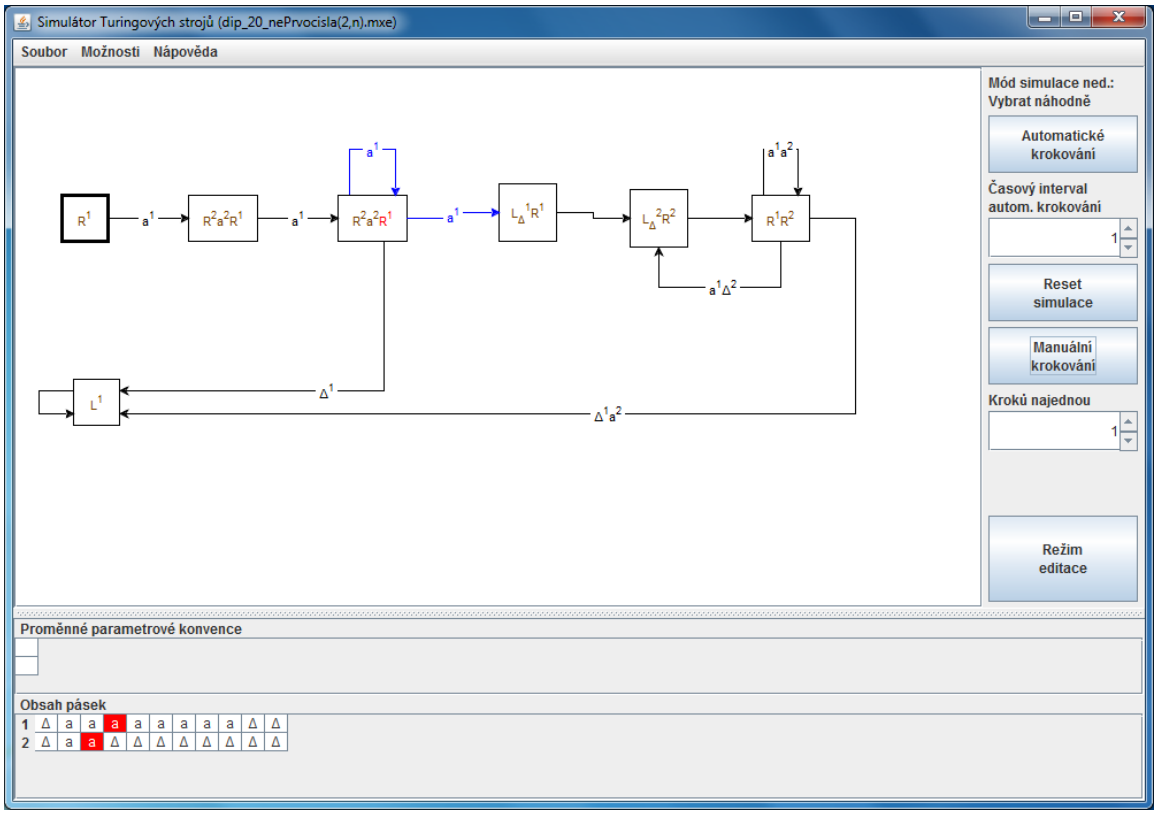
Screenshoty aplikace



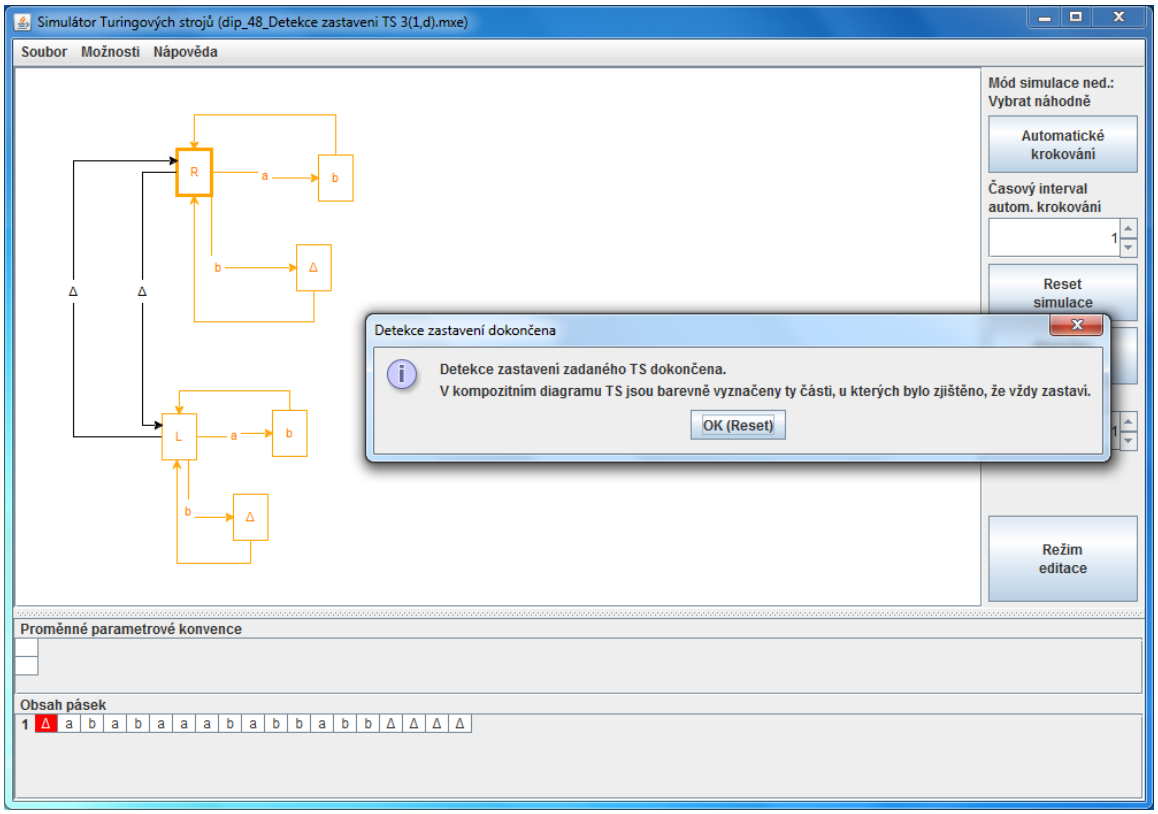
Obrázek C.1: Okno aplikace – Editační režim



Obrázek C.2: Okno aplikace – Simulační režim



Obrázek C.3: Okno aplikace – Simulační režim – Volba následujícího přechodu



Obrázek C.4: Okno aplikace – Simulační režim – Detekce zastavení